Table of Contents

Unique Characters Catalog

$\mathbb{Z}$ - integers
$\mathbb{Z}^+$ - positive integers
$\mathbb{R}$ - real numbers
$\mathbb{N}$ - natural numbers
$\mathbb{Q}$ - rational numbers
$\mathbb{C}$ - complex numbers

$\in$ - is an element of
$\notin$ - is not an element of
$\wedge$ - AND
$\vee$ - OR
$\neg$ - NOT
$\rightarrow$ - implies
$\leftrightarrow$ - biconditional, if and only if
$\equiv$ - is congruent to, is equivalent to

$\lfloor x \rfloor$ - floor (round down)
$\lceil x \rceil$ - ceiling (round up)
$\therefore$ - therefore, thus

$\forall$ - universal quantifier (for all)
$\exists$ - existential quantifier (there exists some)
$\exists!$ - unique existence quantifier (exactly one value exists)

$\top$ - true
$\bot$ - false

$a|b$ - divides
$(mod\ k)$ - multiple of k

⊆ - is a subset of
− - set difference
× - Cartesian product
∪ - union (in at least one set)
∩ - intersection (in both)
∅ - empty set

Week 1: Logic

1.1 Math review

- Types of number sets
  - Z - integers
  - N - non-negative numbers
  - Z^+ - positive integers
  - R - real numbers
  - Q - rational numbers
  - C - complex numbers
- $x \in \mathbb{R}$ - x is an arbitrary real number

1.2 Pairs of reals

- intended meaning affects the operations we can do on the pairs (interval vs 2D point)

1.3 Exponentials and logs

- Refer to rules...
  - $y = b^x$ can turn into $\log_b(y) = x$ if $b > 1$
  - logs appear as runtimes of very fast algorithms
  - log(x) with no explicit base will mean $\log_2(x)$

1.4 Some handy functions

- Factorial - k! = (k-1)k
  - 0! = 1
- A set of n objects has n! different ways of rearranging them into a specific order
  - $\frac{n!}{k!(n-k)!} = nCr$
- The floor function returns the biggest integer that is less than x (rounds down)
  - floor 3.75 = 3
  - floor -3.75 = -4
- The ceiling function does the same thing, but rounds up

1.7 Variations in notation

- Logs in base 2 work nicely in CS problems

2.2 Propositions

- proposition - a statement that is either true or false (never both)
    - cannot include variables so predicate logic is more used here

2.3 Complex propositions

- Statements can be joined together to make more complex statements
    - Each simple statement can be represented by a variable
- $p \wedge q$ - p and q (true when p and q are true)
- $p \vee q$ - p or q (true when p or q are true)
- $\neg p$ - not p (true when p is false)
- $p \oplus q$ - p exclusive or q (true when p and q are different)

2.4 Implication

- $p \rightarrow q$ - if p, then q (false only when p is true and q is false; true otherwise)

2.5 Converse, contrapositive, biconditional

- The converse of $p \rightarrow q$ is $q \rightarrow p$ (false only when p is false and q is true; true otherwise)
    - not always equivalent to the original statement
- The biconditional of $p \rightarrow q$ is $p \leftrightarrow q$ (true when p and q are both true or both false)
- The contrapositive of $p \rightarrow q$ is $\neg q \rightarrow \neg p$
    - always equivalent to the original expression

2.6 Complex statements

- apply "not" operators before "and" and "or"

2.7 Logical Equivalence

- Two propositions are logically equivalent if they are true for the exact same input values
    - represented by $p \equiv q$
- De Morgan's Laws:
    - $\neg(p \wedge q)$ is equivalent to $\neg p \vee \neg q$
    - $\neg(p \vee q)$ is equivalent to $\neg p \wedge \neg q$

2.8 Some useful logical equivalences

- Distribution works differently in logic (distribute statement and operator)
    - $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
    - $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

2.9 Negating propositions

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

2.10 Predicates and Variables

- A predicate is a statement that becomes true or false if you sub in values for its variables
    - $P(x)$ instead of p
- depends on what values x can have (ex: integers vs natural numbers)

2.11 Other quantifiers

- There are three quantifiers in mathematics:
    - "for all" - universal quantifier (all values)
    - "there exists" - existential quantifier (some values)
    - "unique" - existence quantifier (1 value)

2.12 Notation

- $\forall$ - universal quantifier
    - ex: $\forall x \in \mathbb{R},\ x^2 + 3 \geq 0$
- $\exists$ - existential quantifier
    - ex: $\exists x \in \mathbb{R},\ x = \sqrt{2}$
- $\exists!$ - unique existence quantifier
    - ex: $\exists! x \in \mathbb{R},\ x^2 = 0$

2.13 Useful Notation

- transforming statements does NOT change the quantifier
    - original: $\forall$x, if p(x), then q(x)
    - contrapositive: $\forall$x, if ¬q(x), then ¬p(x)

2.14 Notation for 2D points

- point on a (unit) circle: $\exists(x, y) \in \mathbb{R},\ x^2 + y^2 = 1$

2.15 Negating statements with quantifiers

- A predicate statement is proven false if there is at least one value that makes the claim false
    - transformation ex: ¬(∀x, P(x)) ≡ ∃x, ¬P(x)
    - (for all becomes there exists and vice versa with "not" operators)
- example: negating ∀x, P(x) → (Q(x) ∧ R(x)) becomes ∃x, P(x) ∧ (¬Q(x) ∨ ¬R(x))

2.16 Binding and scope

- binding - quantifier linked to a variable
- scope - quantifying the variable for a limited time (usually end of the quantifying statement/line)
- If a variable is not bound, it is "free" and is not a step in a proof since it does not have a defined truth value

2.17 Variations in Notation

- In certain programming languages, "true" and "false" are represented by 1 and 0

8.28.25 - Lecture

- propositional logic - zeroth-order logic
- predicate logic - first-order logic

- Propositions:
    - All statements have truth values (should be declarative)
        - should be consistent values
    - need to be grammatically correct and have semantic meaning
    - The way you form sentences matters (will be graded on that)
- truth values are either true(T) or false (upside down T)
- statements depend on context
- Proposition ex: The sky is blue
    - This is subjective b/c it could be raining, so context is very important
- Another ex: the bearing on that piston is 3 microns too large
    - contains important information that we should be on the same page on
- Sentences are equivalent when they have the same truth value
- Bad ex: "This sentence is false"
    - can have both true and false values at the same time, which can cause problems
- Compound ex: "Espresso is delicious and nutritious"
    - Split into "Espresso is delicious" and "Espresso is nutritious"
- However, not all compound statements end up as false from one false component
    - "Espresso is delicious but not nutritious"
    - The last statement is wrong, turning into true, and making the end statement true
- End statements depend on the logical connectives of the individual components
    - conjunction - $\wedge$ (and, but, also)
    - disjunction - $\vee$ (or) (can be a, b, or both)
    - negation - $\neg$ (not)
    - conditional - $\rightarrow$ (if..., then...)
    - biconditional - $\leftrightarrow$ (if and only if [iff], equivalence)

A	B	A∧B	A∨B	A↔B	A→B
T	T	T	T	T	T
T	⊥	⊥	T	⊥	⊥
⊥	T	⊥	T	⊥	T
⊥	⊥	⊥	⊥	T	T

- These statements exist at the same time and always have the same value
- ex: "Everyone shows up to class"
    - Define what you mean by everyone
- predicates - structures that contain variables so that when all of the variables are replaced by objects, we get a proposition
- ex: S(x) = "X shows up to class"
    - x is an object that comes from a universe of discourse (set)
    - ∀x (S(x))
        - R universe: all humans registered for CS173 at UIUC
        - T universe: all 7-foot humans
        - H universe: all humans who are here
            - $(\forall x \in H)$ (S(x)) [which is true]
        - S universe: all 50-feet humans tall
            - $(\forall x \in S)$ (S(x))
- ex: S(x) = "at least one person shows up to class"
    - $(\exists x \in R)$ (¬S(x)) - there exists one person registered for class who does not show up
    - ¬$(\forall x \in R)$ (S(x)) - it is not the case that, for all people registered for class, not everyone shows up to class
- theorem ex: (∀x) (f(x)) ≡ (¬∃x) (¬f(x))
- another theorem ex: (¬∃x) (f(x)) ≡ (∀x) (¬f(x))
    - you can use helper functions (i.e let g(x) = ¬f(x))
- double negation theorem: if p is a proposition, then p = ¬(¬p)
- De Morgan's Laws:
    - If p and q are propositions, then..
        - ¬(p ∧ q) ≡ ¬p ∨ ¬q
        - ¬(p ∨ q) ≡ ¬p ∧ ¬q

Week 2: Proofs and Number Theory

3.1 Proving a universal statement

- The simplest technique for proving $\forall x \in A, P(x)$ is finding a random element in the set that satisfies the statement

3.2 Another example

- Claim: For any integer *k*, if *k* is odd, then k^2 is odd
- Definition 1: An integer n is even if there is an integer m such that n=2m
- Definition 2: An integer n is odd if there is an integer m such that n=2m+1
- Proof of claim 1: $k^2 = (2j + 1)^2 = 4j^2 + 4j + 1 = 2(2j^2 + 2j) + 1$

3.3 Direct proof outline

- start with known information and move gradually towards the info that needs to be proved

3.4 Proving existential statements

- Claim: There is an integer *k* such that $k^2 = 0$
- Proof: zero is such an integer, so the statement is true
    - Pick your variable rather than a random one in universal proofs

3.5 Disproving a universal statement

- Claim: Every rational number q has a multiplicative inverse
- Definition 1: If q and r are real numbers, r is a multiplicative inverse for q if rq=1
- Disproof: The claim is false because 0 does not have a multiplicative inverse
- Statements like these are false if you can find at least one value that does not make the statement true

3.6 Disproving an existential statement

- The negation of an existential statement is a universal statement
- Claim: There is an integer k such that $k^2 + 2k + 1 < 0$
- Same claim in existential words: For every integer k, it is not the case that $k^2+2k+1<0$
    - In other words: For every integer k, $k^2 + 2k + 1 \geq 0$

3.7 Recap of proof methods

So, our general pattern for selecting the proof type is:

	prove	disprove
universal	general argument	specific counter-example
existential	specific example	general argument

3.8 Direct proof with two variables

- Definition: an integer is a perfect square if $n=k^2$ for some integer k
- Claim: For any integers m and n, if m and n are perfect squares, then so is mn
- Proof: Let m and n be integers that are perfect squares. By definition of a perfect square, we know $m = k^2$ and $n = j^2$. This means $mn=k^2j^2$ which is $(kj)^2$. Since mn is the square of the integer kj, QED.
    - You should use a fresh variable every time you expand on a definition

3.10 Proof by cases

- When given information allows two or more separate possibilities, it is best to use a technique called proof by cases
- Claim: For all integers j and k, if j is even or k is even, then jk is even
- Proof: Let j and k be integers, and suppose that j is even or k is even
    - Case 1: j is even. Then j = 2m where m is an integer. So jk = 2mk. Since m and k are integers, so is mk. So, jk must be even
    - Case 2: k is even. Then k = 2n where n is an integer. So jk = 2nj. Since n and j are integers, so is nj. So, jk must be even.
    - jk is even in both cases, which is what we needed to show

3.11 Rephrasing claims

- Initial claim: There is no integer k such that k is odd and k^2 is even
- Same claim: For every integer k, it is not the case that k is odd and k^2 is even
- Same claim: For every integer k, k is not odd or k^2 is not even
- Same claim: For every integer k, k is not odd or k^2 is odd
- Same claim: For every integer k, if k is odd, then k^2 is odd ($\neg p \vee q$ is equivalent to $p \rightarrow q$)

3.12 Proof by contrapositive

- Reminder: contrapositive is negating the hypothesis and conclusion AND swapping them!
- Claim: For any integers a and b, if a+b≥15, then a>8 or b>8
- Contrapositive claim: a and b, if a<8 and b<8, then a+b<15
- Proof: We'll prove the contrapositive of the statement. Suppose that a and b are integers such that a<8 and b<8. Since they are integers, this implies $a \leq 7$ and $b \leq 7$. Adding this together results in $a + b \leq 14$ which implies $a + b < 15$.

4.1 Factors and multiples

- Definition: Suppose that a and b are integers. Then a divides b if b = an for some integer n. a is a divisor of b. b is a multiple of a.
    - shorthand for a divides b is a | b
    - Examples:
        - 7 | 77 (true)
        - 77 | 7 (false; divisor must be on the left)
        - 7 | 7 (true)
        - 7 | 0 (true)
        - 0 | 7 (false; division by 0)
- An integer p is even exactly when 2 | p

4.2 Direct proof with divisibility

- Claim: For any integers a, b, and c, if a | b and a | c, then a | (b+c)
- Proof: Since a | b, there is an integer k such that b = ak. Similarly, since a | c, there is an integer j such that c = aj. Adding these two equations, we find that b+c=a(k+j). Since k and j are integers, so is k+j. Therefore, by the definition of divides, a | (b+c)

4.3 Stay in the set

- Do NOT rephrase "a | b" as "$\frac{b}{a}$". This introduces a non-integer rational number
- Purely integer proofs are typically simpler
- using rationals to prove facts about integers can lead to circular proofs

4.4 Prime numbers

- Definition: an integer $q \geq 2$ is prime if the only positive factors of q are q and 1. An integer $q \geq 2$ is composite if it is not prime.

4.5 GCD and LCM

- The largest common divisor between two numbers a and b is the GCD.
  - shorthand is gcd(a,b)
- The smallest common multiple between two numbers a and b is the LCM
  - shorthand is $\frac{ab}{gcd(a,b)}$
- If a and b do not share any factors, then gcd(a,b) = 1. Such a pair is called relatively prime

4.6 The division algorithm

- There is a fast way to determine the GCD of larger integers
- Theorem: for any integers a and b, where b is positive, there are unique integers q (quotient) and r (remainder) such that a = bq + r and $0 \leq r < b$
- Corollary - really easy consequence of the preceding claim

4.7 Euclidean algorithm

```
gcd(a,b: positive integers)
   x := a
   y := b
   while (y > 0)
      begin
      r := remainder(x,y)
      x := y
      y := r
      end
   return x
```

-
- Example of algorithm used with 252 and 105 (GCD result should be 21)

x	y	$r = \text{remainder}(x, y)$
105	252	105
252	105	42
105	42	21
42	21	0
21	0	

  -
  - always does bigger number divided by smaller number

4.9 A recursive version of gcd

```
procedure gcd(a,b: positive integers)
r := remainder(a,b)
if (r = 0) return b
else return gcd(b,r)
```

4.10 Congruence mod k

- Two integers are "congruent mod k" if they differ by a multiple of k
  - For example, a 12-hour clock. 17:00 (on military time) is the same as 5:00 with mod 12
- Definition: If k is any positive integer, two integers a and b are congruent mod k [written $a \equiv b \pmod{k}$] if k | (a-b)
- Examples:
  - $3 \equiv 38 \pmod{7}$ (Since 38− 3 = 35.)
  - $-3 \equiv 4 \pmod{7}$ (Since (−3) + 7 = 4.)
  - $-3 \not\equiv 3 \pmod{7}$
  - $-29 \equiv -13 \pmod{8}$ (Since (−13)− (−29) = 16.)

4.11 Proofs with congruence mod k

- Claim: For any integers a, b, c, d, and k, k positive, if $a \equiv b \pmod{k}$ and $c \equiv d \pmod{k}$, then $a + c \equiv b + d \pmod{k}$
- Proof: Let a, b, c, d, and k be integers with k positive. Suppose that $a \equiv b \pmod{k}$ and $c \equiv d \pmod{k}$. Since $a \equiv b \pmod{k}$, k | (a - b) by the definition of congruence mod k. Similarly, $c \equiv d \pmod{k}$, k | (c - d). k | (a − b) + (c − d) so k | (a + c)− (b + d). $a + c \equiv b + d \pmod{k}$

9.2.25 - Lecture

Intro, body, and conclusion are necessary in proofs!

Simple Example:

For any integers a and b, if a and b are odd, then ab is also odd.

Proof:

- Definition: An integer x is odd iff there exists some integer k in the set such that x=2k+1
- Let a and b be odd integers. Since a is odd, $a = 2k + 1$ and similarly, since b is odd, $b = 2j + 1$ where k and j are also integers. $ab = (2k + 1)(2j + 1)$, which results in $4jk + 2k + 2j + 1$. Factoring this results in $2(2jk + k + j) + 1$, the definition of an odd number. Since 2, j, k are integers $ab = 2(some\ int) + 1$.
- ∴ ab is odd by definition.

Proof with divisibility:

For any integers a, x, y, b, c, if a | x and a | y, then a | bx + cy.

Proof:

- Definition: For any integers a and b, a | b iff b = ak for some integer k
- Let a, x, y, b, c be any integers such that a | x and a | y. Since a | x, x = ja and similarly since a | y, y = ka for some integers k and j. $bx + cy = b(ja) + c(ka) = a(bj + ck)$. Since b, j, c, and k are all integers, $bj + ck$ is also an integer. $bx + cy = a(some\ int)$
- ∴ $a \mid bx + cy$

Disproving Existential Statements:

Claim to disprove: There exists a real x, $x^2 - 2x + 1 < 0$

Proof:

- To do this, I will prove the negative, which is $x \in \mathbb{R},\ x^2 - 2x + 1 \geq 0$
- Let x be a real number. Factor the expression to get $(x - 1)^2 \geq 0$.
- Case 1: Consider the case where x < 1. Then, x-1<0 and thus negative, and any negative x a negative is positive.
- Case 2: Otherwise, the case is $x \geq 1$, thus $x - 1 \geq 0$. is also non-negative.
- In both cases, $x^2 - 2x + 1$ is non-negative, therefore $x^2 - 2x + 1 \geq 0$

Disproving Universal Statements:

Claim to disprove: For all real x, $(x + 1)^2 > 0$

Proof:

- To do this, I will prove the negative, which is $\exists x \in \mathbb{R},\ (x + 1)^2 \leq 0$
- Consider x = -1. This causes the $(x + 1)^2$ to equal 0, which is not >0.
- ∴ The claim is false.

Proof with Modulus:

For any integers a, b, c, d, k with k > 0, if a ≡ b (mod k) and c ≡ d (mod k) then (a + c) ≡ (b + d) (mod k).

Proof:

- Definition: Given some integers a and b, and a positive integer k, a ≡ b (mod k) iff $k \mid a - b$
- Let a, b, c, d, k be some integers such that k > 0, a ≡ b (mod k), and c ≡ d (mod k). Since a ≡ b (mod k), k | (a-b), and similarly, since c ≡ d (mod k), k | (c-d) by definition of congruence mod k. This means k | (a-b)+(c-d), which simplifies to k | (a+c)-(b+d), the definition of congruence mod k.
- ∴ (a + c) ≡ (b + d) (mod k)

Week 3: Set Theory

4.12 Equivalence classes

- equivalence class - a group of congruent integers treated as a unit
    - known as integers mod k

5.1 Sets

- A set is an unordered collection of objects
    - Items in a set are called elements
- Three ways to define a set:
    - Describe its contents in mathematical English
    - List all its members
    - Use set builder notation

5.2 Things to be careful about

- Ex: {1,2,3} and {2,3,1} are two names for the same set
- Ex: {1,2,3} and {1,2,3,2} also name the same set
- An ordered sequence of k numbers is a k-tuple
- Tuples are different from sets because order matters, and duplicate elements do not collapse
    - ex: (1,2,3) ≠ (1,2,2,3)
- Tuples also need at least two elements
- Sets and tuples can contain objects of more than one type

5.3 Cardinality

- cardinality - if A is a finite set, then |A| is the number of different objects in A
- If A and B are sets, then A is a subset of B (written $A \subseteq B$) if every element of A is also in B
    - $A \subseteq A$ is true for any set A
- If the two sets are different, then A must be a proper subset of B (written $A \subset B$)
- $B \supseteq A$ means the same as $A \subseteq B$

5.4 Vacuous truth

- vacuously true - only true because of a strange convention about the meaning of conditionals
- empty sets are technically subsets of (for example) set A

5.5 Set operations

- Given two sets A and B, the intersection of A and B ($A \cap B$) is the set containing all objects that are in both A and B.
- The union of sets A and B ($A \cup B$) is the set containing all objects that are in one (or both) of A and B.
- The set difference of A and B ($A - B$) contains all the objects that are in A but not in B.
- The complement of a set A (A bar) contains all objects that aren't in A
- If A and B are two sets, their Cartesian product ($A \times B$) contains all ordered pairs (x, y) where x is in A and y is in B.
    - order matters in Cartesian products
    - ex: if A = {a,b} and B = {1,2}, then A x B = {(a,1), (a,2), (b,1), (b,2)}

5.6 Set identities

- DeMorgan's Law: $\overline{A \cup B} = \overline{A} \cap \overline{B}$

5.7 Size of set union

- The size of the union of set A and B is the sum of their individual sizes
    - $|A \cup B| = |A| + |B|$
- Inclusion-Exclusion Principle: $|A \cup B| = |A| + |B| - |A \cap B|$

5.8 Product rule

- if you have p choices for one part of a task, then q choices for a second part, and your options for the second part don't depend on what you chose for the first part, then you have pq options for the whole task.

5.10 Proving facts about set inclusion

- Proofs typically involve reasoning about subset relations, even when proving two sets to be equal.
- Ex: proving A is a subset of B
    - Proof: Let sets A and B be defined as above. Let x be an element of A. Then $x = \mu(2, 3) + (1- \mu)(7, 4)$ for some $\mu \in [0, 1]$. So $x = (p, q)$ where $p = 2\mu + 7(1- \mu)$ and $q = 3\mu + 4(1- \mu)$
    - Simplifying these equations, we get that $p = 2\mu + 7- 7\mu = 7- 5\mu$ and $q = 3\mu + 4- 4\mu = 4- \mu$ Since μ is in the interval $[0, 1]$, $\mu \leq 1$. So $7- 5\mu$ and $4- \mu$ are both ≥ 0. So $p \geq 0$ and $q \geq 0$. This means that $x = (p, q)$ is an element of B.
    - Since x was arbitrarily chosen, we've shown that any element of A is also an element of B. So A is a subset of B.

5.11 An abstract example

- Claim: For any sets A, B, and C, if $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$
- Proof: Let A, B, and C be sets and suppose that $A \subseteq B$ and $B \subseteq C$.
- Let x be an element of A. Since $A \subseteq B$ and $x \in A$, then $x \in B$ (definition of subset). Similarly, since $x \in B$ and $B \subseteq C$, $x \in C$. So for any x, if $x \in A$, then $x \in C$. So $A \subseteq C$ (definition of subset again).

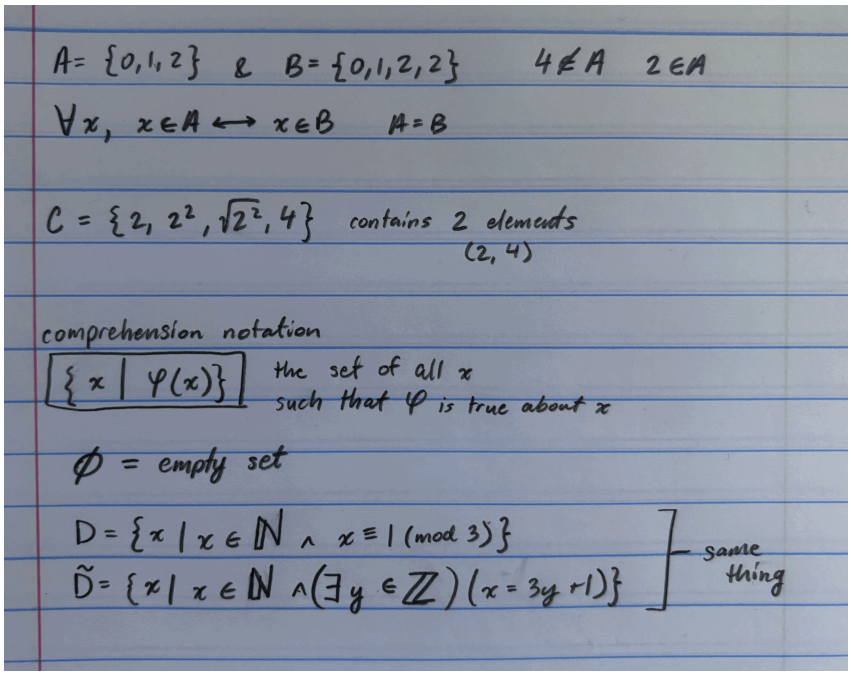
5.12 An example with products

- Claim: For any sets A, B, and C, if $A \times B \subseteq A \times C$ and $A \neq \varnothing$, then $B \subseteq C$.
- Proof: Suppose that A, B, and C are sets and suppose that $A \times B \subseteq A \times C$ and $A \neq \varnothing$. We need to show that $B \subseteq C$. So let's choose some $x \in B$. Since $A \neq \varnothing$, we can choose an element t from A. Then $(t, x) \in A \times B$ by the definition of Cartesian product.
- Since $(t, x) \in A \times B$ and $A \times B \subseteq A \times C$, we must have that $(t, x) \in A \times C$ (by the definition of subset). But then (again by the definition of Cartesian product) $x \in C$. So we've shown that if $x \in B$, then $x \in C$. So $B \subseteq C$, which is what we needed to show.

5.13 A proof using sets and contrapositive

- Claim: For any sets A and B, if $(A - B) \cup (B - A) = A \cup B$ then $A \cap B = \varnothing$.
- Contrapositive claim: For any sets A and B, if $A \cap B \neq \varnothing$, then $(A - B) \cup (B - A) \neq A \cup B$
- let A and B be sets and suppose that $A \cap B \neq \varnothing$. Since $A \cap B \neq \varnothing$, we can choose an element from $A \cap B$. Let's call it x. Since x is in $A \cap B$, x is in both A and B. So x is in $A \cup B$. However, since x is in B, x is not in $A - B$. Similarly, since x is in A, x is not in $B - A$. So x is not a member of $(A - B) \cup (B - A)$.
- This means that $(A - B) \cup (B - A)$ and $A \cup B$ cannot be equal, because x is in the second set but not in the first

9.9.25 - Lecture

- Sets are collections of objects (from a universe of discourse)
    - {0,1,2} is a set
    - 0,1,2 are elements
- Basic characteristics of sets
    - Orders in sets do not matter
    - Repetition does not matter
- ø means empty set
- D = {x | x is N & exists y is N such that (x ≡ 3y+1)}
- 

- Example:
    - Let A = {x ∈ N | exists n in N such that x = 2^n and n > 1}
    - Let B = {x ∈ Z | x ≡ 0 (mod 2)}
    - Claim: A ⊆ B
    - Def: set A is a subset of another set B iff for all x (x ∈ A arrow x ∈ B)
    - Proof: Let sets A and B be defined as above. Let x ∈ A. Since x ∈ A, we know x is a natural number, and there is some n in natural numbers such that x = 2^n and n>1. Observe that x = 2^n = 2(2^n-1). Since n-1>0 and n-1 ∈ Z, we can see that 2^n-1∈ Z. Therefore, by definition, 2 | x, which is equivalent to 2 | x-0. Therefore, x ≡ 0 (mod

2) by definition, and since we know $x \in Z$ because $x \in N$, we know $x \in B$. Therefore, $A \subseteq B$. QED

- – means set difference
- x means Cartesian product
- Example:
    - Let A, B, and C be sets. Show that $(A-B) \times C \subseteq (A \times C)-(B \times C)$.
    - Def: $S \times T = \{(a,b) \mid a \in S \text{ and } b \in T\}$
    - Def: $S - T = \{z \mid z \in S \text{ and } z \not\in T\}$
    - Proof: Let A, B, C be arbitrary sets. Let $x \in (A-B) \times C$. So, we know $x = (s,t)$ where $s \in A-B$ and $t \in C$ by definition. We know, then, that $s \in A$ and s not$\in B$. Since $s \in A$ and $t \in C$, then $(s,t) \in A \times C$ by definition. However, since $s \not\in B$ and $t \in C$, then $(s,t) \not\in B \times C$. Since $(s,t) \in A \times C$ and $(s,t) \not\in B \times C$, by definition, this means that $(s,t) \in (A \times C) - (B \times C)$, which is what we needed to prove. Thus, $(A-B) \times C \subseteq (A \times C)-(B \times C)$. QED
- Example:
    - Let $A = \{x \in Z \mid x \equiv 0 \pmod 2\}$ and $B = \{x \in Z \mid x \equiv 0 \pmod 4\}$. Disprove $A \subseteq B$.
    - Consider the integer $x = 2$ and observe that $2 = 2(1)$ and $1 \in Z$, so $2 \mid 2$. This means $2 \mid 2-0$ because $2-0 = 2$. $2 \equiv 0 \pmod 2$ by definition. This tells us that $2 \in A$.
    - Further, we know $4 \not\mid 2$ because $|4| > |2|$. Therefore, 2 is not congr 0 (mod 4). This tells us $2 \not\in B$.
    - So, $A \not\subseteq B$ QED.

Week 4: Set Collections

18.1 Sets containing sets

- Sets containing sets arise naturally when you need to consider multiple subsets of a base set
- We can also construct a set of overlapping groups
- Imagine sets of sets as boxes (ex: one box of three boxes)

18.2 Powerset and set-valued functions

- If A is a set, the powerset of A (written $P(A)$) is the collection containing all subsets of A
    - There are 2^n elements in P(A)
- $P(\emptyset) = \{\emptyset\}$
    - There are 2 elements since {∅} has 1 element
    - This is common in every powerset **NOT ∅ BY ITSELF**
- Type signature: $f : A \to P(A)$

18.3 Partitions

- When a base set A is divided into non-overlapping subsets that include every element of A, the result is a partition of A
- Must satisfy 3 conditions:
    - Covers all of A ($A_1 \cup A_2 \cup \ldots \cup A_n = A$)
    - A is non-empty
    - There is no overlap

18.4 Combinations

- A subset of size k is called a k-combination
- We have $\frac{n!}{(n-k)!}$ ways to choose k elements in some particular order

18.6 Combinations with repetition

- We are picking from a group of objects k from a list of n types. The number of possible combinations is $(\frac{k+n-1}{n-1})$
- Ex: If we wanted to pick 20 plants and there are 5 types available, we would have ${}_{24}C_{20}$ options for how to make the selection

18.7 Identities for binomial coefficients

- Definition of n choose k: $\binom{n}{k} = \binom{n}{n-k}$
- Pascal's Identity: $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$
    - $\binom{n}{0} = \binom{n}{n} = 1$

18.8 Binomial Theorem

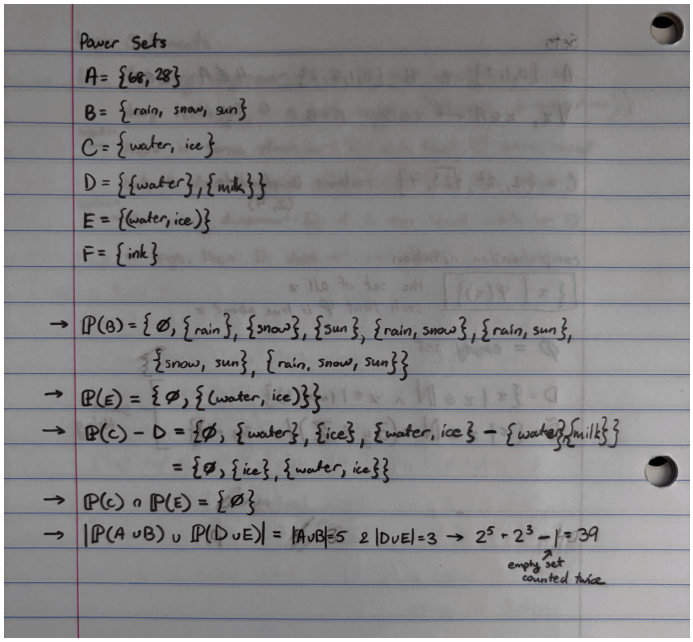
- $(x + y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$

18.9 Variation in notation

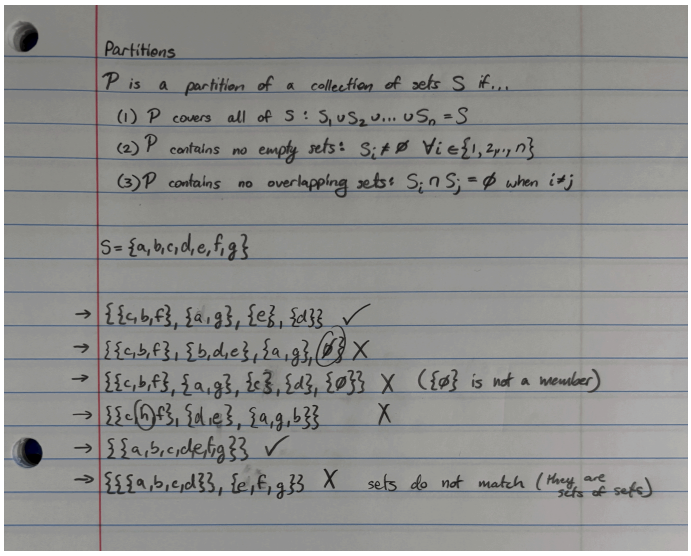
- P(A) can also be represented as 2^A
- A collection is a set of sets (like a 2D array)

9.16.25 - Lecture

- Power Sets:



- Partitions:



- Combinations:
  - You need to form a battle group of 11 made up of orcs, elves, and goblins. In how many ways can you choose the composition of your battle group?
    - xxx|xxxxx|xxx → (n + k - 1) choose (k - 1)
    - A: $(13C2)$ [13 choose 2]

- You need to form a battle group consisting of an orc, an elf, and a goblin whose total strength is 12. The strength of each creature is an integer between 1 and 10, and the strength of the group is the sum of the individual strengths. In how many ways can you construct your battle group?
    - xxx xxx||xxxxxx (set 3 aside for 1 being the lowest strength for all three groups)
    - A: $11C2$ [11 choose 2]

Week 5: Functions

7.1 Functions

- A function f from A to B is an assignment of exactly one element of B
  - A is called the domain of f, and B is called the co-domain
  - Can be written as $f : A \rightarrow B$ (known as a type signature)

7.2 When are functions equal?

- Two functions are equal when they have the same type signature and assign the same input values to output values
- The following functions are the same
  - f : Z → Z such that f (x) = |x|.
  - f : Z → Z such that f (x) = max(x,−x).
  - f : Z → Z such that f (x) = x if x ≥ 0 and f (x) =−x if x ≤ 0.

7.3 What isn't a function?

- A function must only provide one and only one output value

7.4 Images and onto

- The image of a function $f : A \rightarrow B$ is the set of values produced when *f* is applied to all elements of A. Basically, $f(A) = \{f(x) : x \in A\}$
  - Example: if M = {a, b, c, d} and N = {1, 2, 3, 4} and the function $g : M \rightarrow N$ does not have 2 as an output, then $g(A) = \{1, 3, 4\}$
- A function $f : A \rightarrow B$ is onto if its image is its whole co-domain. Basically, $\forall y \in B,\ \exists x \in A,\ f(x) = y$
  - The earlier example is not onto since no input value is mapped to 2

7.6 Negating onto

- First example with negating two quantifiers
- By negating $\forall y \in B,\ \exists x \in A,\ f(x) = y$, we get $\exists y \in B,\ \forall x \in A,\ f(x) \neq y$
- In other words, to show f is not onto, you must find some value y in B, such that for any x in A, f(x) is NOT equal to y

7.7 Nested quantifiers

- Placement of the quantifiers matters in a statement because they can completely change the interpretation of the statement
    - This issue will only happen if there is more than one type of quantifier in the statement (If all the quantifiers are the same, the statement's meaning will not change)

7.8 Proving a function is onto

- Claim: Define the function g from the integers to the integers by the formula $g(x) = x - 8$. g is onto.
- Proof: We need to show that for every integer y, there is an integer x such that g(x) = y. So, let y be some integer. Let x = (y + 8). x is an integer since it is the sum of two integers. Then, $g(x) = (y + 8) - 8 = y$. We have found the required pre-image for y, so the proof is done.

7.9 A 2D example

- Claim: Let $f : \mathbb{Z}^2 \to \mathbb{Z}$ be defined by $f(x, y) = x + y$. Prove f is onto.
- Proof: Let y be an element of $\mathbb{Z}$. Then (0, y) is an element of $f : \mathbb{Z}^2$ and $f(0, y) = 0 + y = y$. Since this setup will work with any choice of y, we have shown that f is onto.

7.10 Composing two functions

- Suppose that $f : A \to B$ and $g : B \to C$ are functions. Then $g \circ f$ is the function from A to C is defined by $(g \circ f)(x) = g(f(x))$.

7.11 A proof involving composition

- Claim: For any sets A, B, and C, and for any functions $f : A \to B$ and $g : B \to C$, if f and g are onto, then $g \circ f$ is also onto.
- Proof: Let A, B, and C be sets. Let $f : A \to B$ and $g : B \to C$ be functions. Suppose f and g are onto.
  We need to show that for any element x in C, there is an element y in A such that $(g \circ f)(x) = x$. So, pick some element x in C. Since g is onto, there is an element z in B such that $g(z) = x$. Since f is onto, there is an element y in A such that $f(y) = z$. Substituting the value $f(y) = z$ into the equation $g(z) = x$, we get $g(f(y)) = x$. So y is the element of A we needed to find

8.1 One-to-one

- Suppose that $f : A \to B$ is a function from A to B. If we pick a value $y \in B$, then $x \in A$ is a *pre-image* of y if $f(x) = y$
    - y might have multiple pre-images
- A function is one-to-one if each output has only one pre-image
- Formal definition of one-to-one:
    - $\forall x, y \in A, f(x) = f(y) \to x = y$

8.2 Bijections

- A function that is both onto and one-to-one is called a bijection

8.3 Pigeonhole Principle

- The Pigeonhole Principle: Suppose you have n objects and assign k labels to these objects. If n > k, then two objects must get the same label.
- Example: If you have 8 playing cards and the cards come in 5 colors, then at least two of the cards will share a color

8.4 Permutations

- An arrangement of n objects in order is called a permutation of n objects
- There are $\frac{n!}{(n-k)!}$ different k-permutations of n objects

8.5 Further applications of permutations

- Example: the number of reorderings of J = (a,p,p,l,e,t,r,e,e,s) is $\frac{10!}{2!3!}$
    - Divide out the number of ways we can permute the duplicates
    - In this example, there are two p's and three e's

8.6 Proving that a function is one-to-one

- Claim: Let $f : \mathbb{Z} \to \mathbb{Z}$ be defined by $f(x) = 3x + 7$. f is one-to-one.
- Proof: Let x and y be integers and suppose that f(x) = f(y). We need to show x = y. Since we know f(x) = f(y), by substitution, $3x + 7 = 3y + 7$, which means x = y by basic algebra. This is what we needed to show.

8.7 Composition and one-to-one

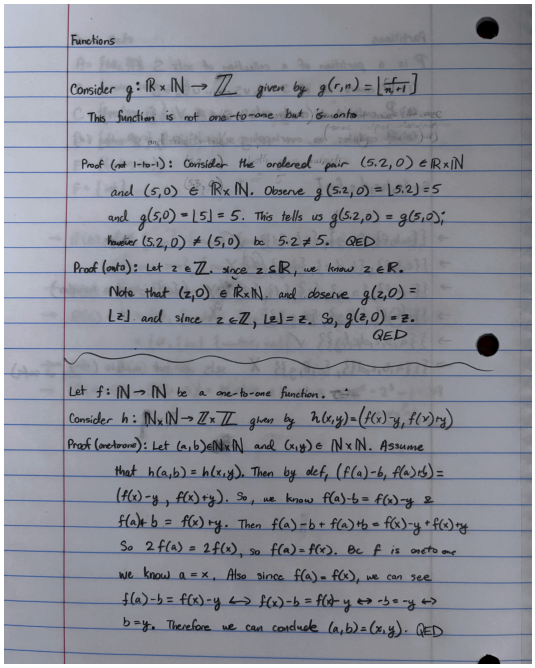
- Claim: For any sets A, B, and C and for any functions $f : A \to B$ and $g : B \to C$, if f and g are one-to-one, then $g \circ f$ is also one-to-one.
- Proof: Let A, B, and C be sets. Let $f : A \to B$ and $g : B \to C$ be functions. Suppose that f and g are one-to-one. We need to show that $g \circ f$ is one-to-one. So, choose x and y in A and suppose that $(g \circ f)(x) = (g \circ f)(y)$. Using the definition of function composition, we can rewrite this as $g(f(x)) = g(f(y))$.
  Combining this with the fact that g is one-to-one, we find that $f(x) = f(y)$. But, since f is one-to-one, this implies that x = y, which is what we needed to show.

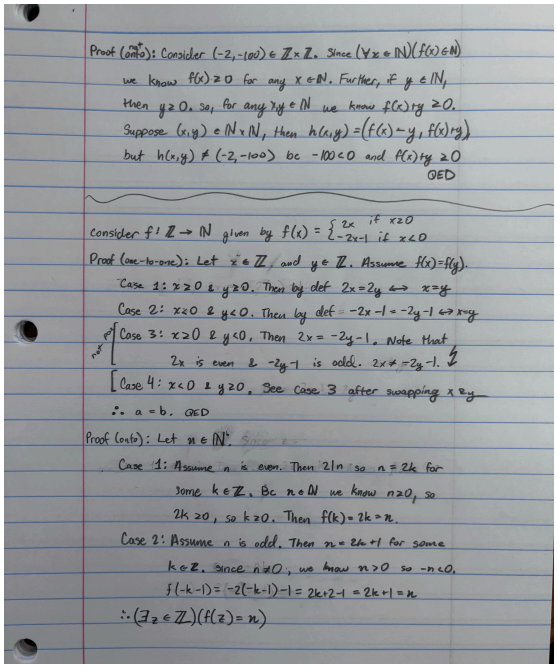
8.8 Strictly increasing functions are one-to-one

- A function $f : A \to B$ is increasing if, for every x and y in A, $x \leq y$ implies that $f(x) \leq f(y)$
  - A similar claim can apply for strictly decreasing functions
- Claim: For any sets of real numbers A and B, if f is any strictly increasing function from A to B, then f is one-to-one.
- Proof: Let A and B be sets of numbers and let $f : A \to B$ be a strictly increasing function. Let x and y be distinct elements of A. We need to show that $f(x) \neq f(y)$.
  Case 1: x < y. Since f is strictly increasing, this implies that $f(x) < f(y)$. So, $f(x) \neq f(y)$
  Case 2: x > y. Since f is strictly increasing, this implies that $f(y) < f(x)$. So, $f(x) \neq f(y)$
  In either case, $f(x) \neq f(y)$, which is what we needed to prove

9.23.25 - Lecture

- Domain - set of inputs
- Co-domain - set of possible outputs
    - Changing the domain and co-domain is enough to change a function entirely
- One-to-one (injective) - different inputs get to different outputs
- Onto (surjective) - the domain A covers the entire co-domain B
- Examples:

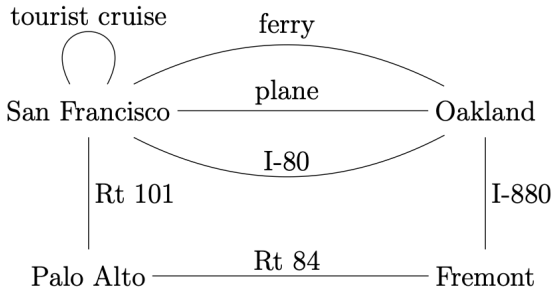




Week 6: Graphs and 2-Way Bounding

9.1 Graphs

- A graph is a set of nodes and a set of edges
    - Commonly referred to as a pair of sets (V, E)
    - Nodes are sometimes called vertices
- Two nodes connected by an edge are called neighbors or adjacent
- A graph can be traversed in both directions (edges are undirected)
- Two nodes can also have multiple edges
- A loop edge is an edge that connects a node to itself
- Graph example:



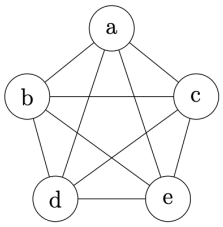
- A graph is a simple graph if it does not have multiple edges or loop edges

9.2 Degrees

- The degree of a node *v* is the number of edges which *v* has an endpoint
    - Written as $deg(v)$
    - If self-loops are allowed, they are counted twice
- Handshaking Theorem: the sum of the degrees of all the nodes is twice the number of edges
    - $\sum_{v \in V} deg(v) = 2|E|$
    - This is true because each edge contributes to two node degrees

9.3 Complete graphs

- A complete graph is one where every node in the graph is connected to every other node in the graph

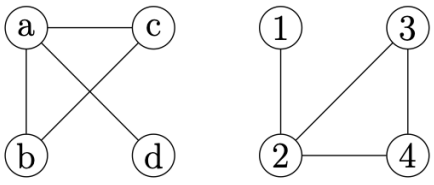


9.4 Cycle graphs and wheels

- A cycle graph is the graph connecting one node to another, with an additional edge connecting the last node to the first node
    - Example: telephone game
    - Cycle graphs have the same number of nodes and edges
- The wheel is a cycle graph with a central node hub that connects to all the others
    - Contains n+1 nodes and 2n edges (where n is the number of nodes in a cycle graph)

9.5 Isomorphism

- An isomorphism from G_1 to G_2 is a bijection $f : V_1 \to V_2$ such that nodes a and b are joined by an edge if and only if $f(a)$ and $f(b)$ are joined by an edge



- Graph properties in isomorphism:
    - The two graphs must have the same number of nodes and edges
    - For any node degree k, the two graphs must have the same number of nodes of degree k

9.6 Subgraphs

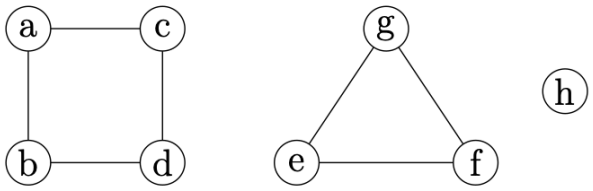
- If G and G' are graphs, then G' is a subgraph of G iff the nodes of G' are a subset of the nodes of G, and the edges of G' are a subset of the edges of G
- If two graphs G and F are isomorphic, then any subgraph of G must have a matching subgraph somewhere in F

9.7 Walks, paths, and cycles

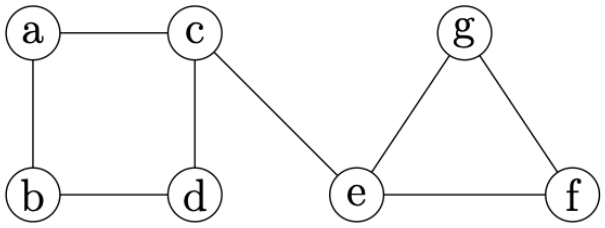
- A walk is a finite sequence of nodes from some node a to another node b and a finite set of edges that connects node a, node b, and every node in between
    - The length of a walk is the number of edges in it
    - The shortest walk is a single node with no edges
- A walk is closed if its starting and ending nodes are the same
    - Closed walks are allowed to reuse nodes
- A path is a walk where no node is used more than once
- A cycle is a closed walk with at least three nodes in which no node is used more than once (excluding the starting and ending nodes, which are the same!)
    - A graph is acyclic if it does not contain any cycles
- A cycle graph contains 2n different cycles (n cycles moving forward, n cycles moving backwards)

9.8 Connectivity

- A graph is connected if there is a walk between every pair of nodes in G
- If we have a graph G that might or might not be connected, we can divide G into connected components
    - Components have the largest possible number of nodes connected to one another, plus the edges



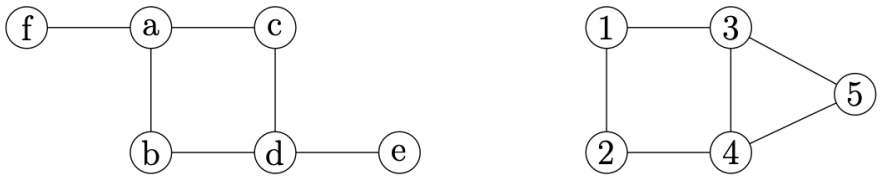
- In this example, there are three connected components
- A cut edge is an edge that, if removed, would cause a graph to become disconnected



- In this case, ce is a cut edge

9.9 Distances

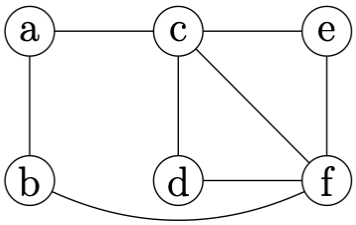
- The distance from node a to node b is the length of the shortest path from a to b
    - Written $d(a, b)$
- Diameter is the maximum distance between any pair of nodes
    - Diameter = $\lfloor n/2 \rfloor$



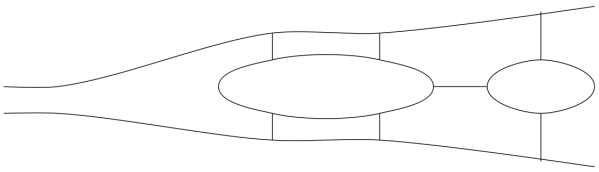
- In this example, $d(f, e) = 4$ and $d(1, 5) = 2$

9.10 Euler circuits

- An Euler circuit is a closed walk that uses each edge exactly once

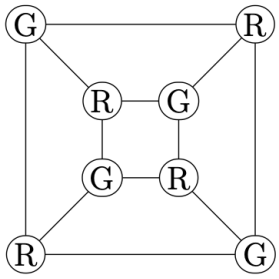


- An Euler circuit is only possible when the graph is connected, and each node has a degree
    - Each node must have an even degree, or else a node will be unused
- Famous example of Euler circuits (Euler proved why it is not possible)

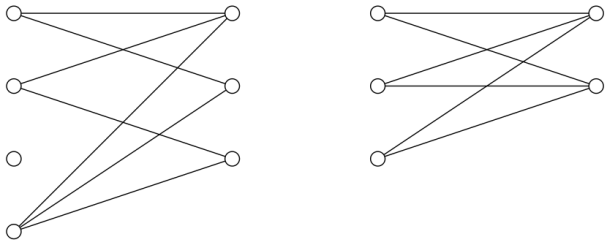


9.11 Bipartite graphs

- A graph is bipartite if we can split the nodes into two non-overlapping subsets such that every edge in G connects an element from the first subset to the second subset



- The complete bipartite graph is a bipartite graph with m nodes in the first subset, n nodes in the second subset, and contains all possible edges that are consistent with the definition of bipartite
    - Has m+n nodes and mn edges



- In this example, the partial bipartite graph is on the left, and the complete bipartite graph is on the right

10.2 Pigeonhole Placement

- Claim: Suppose that T is an equilateral triangle with sides of length 2 units. We can place a maximum of four points in the triangle such that every pair of points is more than 1 unit apart.
- Proof: To show that the maximum is at least four, notice that we can place three points at the corners of the triangle and one point in the center. The points at the corners are two units apart. To see that the point in the center is more than one unit from any corner, notice that the center, the corner, and the midpoint of the side form a right triangle.
- The hypotenuse of this triangle connects the center point to the corner point. Since one leg of the triangle has length 1, the hypotenuse must have length greater than 1
- Suppose we tried to place five or more points into the big triangle. Since there are only four small triangles, by the pigeonhole principle, some small triangle would have to contain at least two points. But since the small triangle has a side length of only 1, these points can't be separated by more than one unit

10.3 Graph coloring

- A coloring of a graph assigns a color to each node in the graph, with the restriction that two adjacent nodes never have the same color
- To establish n as a chromatic number for the graph G:
  - $\chi(G) \leq n$: G can be colored with n colors.
  - $\chi(G) \geq n$: G cannot be colored with fewer than n colors

10.5 Proving set equality

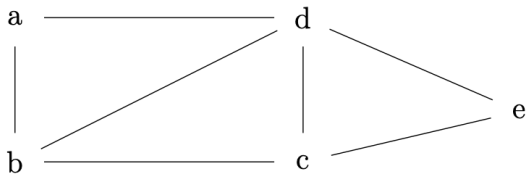
- Claim: Let $A = \{15p + 9q \mid p, q \in \mathbb{Z}\}$. Then $A = \{multiples\ of\ 3\}$
- Proof:
  (1) Show that $A \subseteq \{multiples\ of\ 3\}$. Let $x \in A$. By the definition of A, $x = 15s + 9t$, for some integers s and t. But then $x = 3(5s + 3t)$. 5s + 3t is an integer, since s and t are integers. So x is a multiple of 3.
  (2) Show that $\{multiples\ of\ 3\} \subseteq A$. Notice that (*) 15· (–1) + 9· 2 = 3. Let x be a multiple of 3. Then $x = 3n$ for some integer n. Substituting (*) into this equation, we get $x = (15 \cdot (-1) + 9 \cdot 2)n$. So $x = 15 \cdot (-n) + 9 \cdot (2n)$. So x is an element of A.
  Since we've shown that $A \subseteq \{multiples\ of\ 3\}$ and $\{multiples\ of\ 3\} \subseteq A$, we can conclude that $A = \{multiples\ of\ 3\}$

9.30.25 - Lecture

- Paths only use each vertex once (no repetition)
- Walks allow the reuse of vertices
- Example:

Paths
List all the paths from b to e in graph G below.

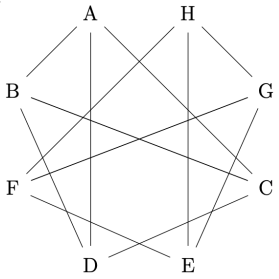
G



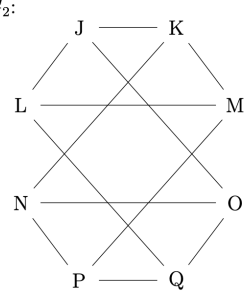
  - (b,d,e), (b,c,e), (b,d,c,e), (b,c,d,e), (b,a,d,e), (b,a,d,c,e)
- Example:

Component Is each of these graphs connected? If not, list the nodes in each connected component.

G_1:



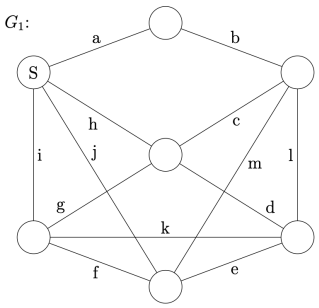
G_2:



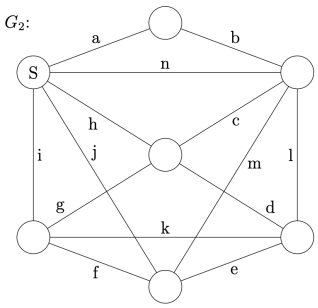
  - $G_1 = 2$ components: (A, B, C, D), (E, F, G, H)
  - $G_2 = 1$ component
- Example:

Euler Circuits Find an Euler circuit in each graph beginning at S, or explain why this isn't possible.

G_1:



G_2:



- G_1 is an Euler circuit (even degree and connected)
  - a,b,l,k,i,h,g,f,e,d,c,m,j
- G_2 is not (S node has an odd degree)

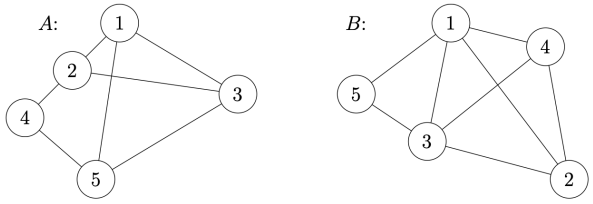
- In general, the obvious upper bound for a graph is the number of nodes, and the obvious lower bound is 1 (since all graphs will have at least one node)
  - However, this will not always be the case for the lower bound
- Example:

Chromatic Number

Recall that the justification that a particular chromatic number is valid requires bounding the number from above *and* below. Therefore you must give an *explicit* coloring to produce an upper bound *and* produce a valid argument that no smaller number of colors will work to produce a lower bound.

The argument justifying the lower bound often involves finding a copy of K_n (where n is the chromatic number you are attempting to validate) as a subgraph. Sometimes, however, you have to work through the space of possible $n-1$ colorings by hand and show that none of them work.

Find and justify the chromatic numbers for each of the following graphs.



  - A bound is 3
    - Need 3 colors to color subgraph (1,2,3)
    - 4 and 5 can reuse colors
  - B bound is 4
    - Need 4 colors to color the subgraph (1,2,3,4)
    - 5 can reuse a color

Week 7: Induction

1.5 Summations

- $\sum_{i=1}^{n} a_i = a_1 + a_2 + \ \ldots \ + a_n$
- Products can be written in a similar notation
  - $\prod_{i=1}^{n} 2^i = 2^1 \bullet 2^2 \bullet \ \ldots \ \bullet 2^n$
- Certain sums can be re-expressed. These are typically geometric series.
  - General formula (where $r \neq 1$): $\sum_{k=0}^{n} r^k = \frac{r^{n+1}-1}{r-1}$
- Summation of integers formula: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

1.6 Strings

- A string is a finite-length sequence of characters, and its length is the number of characters
- The special symbol ϱ is used for the string containing no characters, which has a length of 0
- Concatenation is represented by writing strings next to each other
  - If α=blue and β=cat, then αβ=bluecats
  - βs=cats
- A bit string consists of the characters 0 and 1
- If A is a set of characters, then A* is the set of all finite-length strings containing only characters from A; it also contains the empty string ϱ
- Shorthand notation:
  - $a \mid b$ means either of the characters a and b
  - a* means zero or more copies of the character a

11.1 Introduction to induction

- Induction is a new proof technique
  - Mathematical induction is a technique for showing that a statement P(n) is true for all natural numbers n, or for some infinite subset of the natural numbers
  - Best for analyzing recursive algorithm performance

11.2 Example of induction

- The induction part of the proof is a conditional statement
    - The assumption is called the inductive hypothesis
- Claim: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ is true for all positive integers n.

  Proof: Use induction on n.

  Base: We need to show that the formula holds for n = 1. $\sum_{i=1}^{1} i = 1$ and $\frac{1(2)}{2} = 1$. So the two are equal for n = 1

  Induction: Suppose $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for n = 1, 2, … , k - 1. We need to show that $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$. By the definition of summation notation, $\sum_{i=1}^{k} i = (\sum_{i=1}^{k-1} i) + k$.

  The inductive hypothesis states that at n = k - 1, $\sum_{i=1}^{k-1} i = \frac{(k-1)k}{2}$. Combining these two formulas, we get $\sum_{i=1}^{k} i = \frac{(k-1)k}{2} + k = \frac{k(k+1)}{2}$, which is what we needed to prove.

11.3 Why is this legit?

- Domino Theory: Imagine an infinite line of dominoes. The base step pushes the first one over. The inductive step claims that one domino falling down will push over the next domino in the line. So dominos will start to fall from the beginning all the way down the line. This process continues forever, because the line is infinitely long. However, if you focus on any specific domino, it falls after some specific finite delay.

11.4 Building an inductive proof

- P(n) must be a statement and must depend on an integer n (induction variable)
- Make sure to fill the middle part of the induction step

11.5 Another example

- Claim: For any natural number n, $n^3 - n$ is divisible by 3
  Proof: Use induction on n

  Base: Let n = 0. Then $n^3 - n$ evaluates to 0, which is divisible by any number.

  Induction: Suppose that $n^3 - n$ is divisible by 3 for n = 0, 1, … , k. We need to show that $(k + 1)^3 - (k + 1)$ is divisible by 3.

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

  From the inductive hypothesis, $(k^3 - k)$ is divisible by 3. And $3(k^2 + k)$ is divisible by 3 because $(k^2 + k)$ is an integer. So, $(k + 1)^3 - (k + 1)$ is divisible by 3.
- As long as claim is proven for the next largest integer, it does not matter if the hypothesis goes through n = k - 1 or n = k

11.6 Comments by style

- Mention that you are doing a proof by induction if applicable
- Base case should be short since it is usually fairly obvious
- Label the base and inductive steps
- Mention what you need to prove at the start of the inductive step

11.7 A geometric example

- Claim: For any positive integer n, a $2^n \times 2^n$ checkerboard with any one square removed can be tiled using right triominoes

  Proof: by induction on n.

  Base: Suppose $n = 1$. Then our $2^n \times 2^n$ checkerboard with one square removed is exactly one right triomino.

  Induction: Suppose that the claim is true for $n = 1, \ldots, k$. That is a $2^n \times 2^n$ checkerboard with any one square removed can be tiled using right triominoes as long as $n \leq k$.

  Suppose we have a $2^{k+1} \times 2^{k+1}$ checkerboard C with any one square removed. We can divide C into four $2^k \times 2^k$ sub-checkerboards P, Q, R, and S. One of these sub-checkerboards is already missing a square. Suppose without loss of generality that this one is S. Place a single right triomino in the middle of C so it covers one square on each of P, Q, and R.

  Now look at the areas remaining to be covered. In each of the sub-checkerboards, exactly one square is missing (S) or already covered (P, Q, and R). So, by our inductive hypothesis, each of these sub-checkerboards minus one square can be tiled with right triominoes. Combining these four tilings with the triomino we put in the middle, we get a tiling for the whole of the larger checkerboard C. This is what we needed to construct.

11.8 Graph coloring

- Claim: For any positive integer D, if all nodes in a graph G have a degree ≤ D, then G can be colored with D + 1 colors

  Proof: Let's pick a positive integer D and prove the claim by induction on the number of nodes in G.

  Base: Since $D \geq 1$, the graph with just one node can obviously be colored with $D + 1$ colors.

  Induction: Suppose that any graph with at most $k - 1$ nodes and maximum node degree $\leq D$ can be colored with $D + 1$ colors.

  Let G be a graph with k nodes and maximum node degree $\leq D$. Remove some node v (and its edges) from G to create a smaller graph G'.

  G' has $k - 1$ nodes. Also, the maximum node degree of G' is no larger than D, because removing a node can't increase the degree. So, by the inductive hypothesis, G' can be colored with $D + 1$ colors.

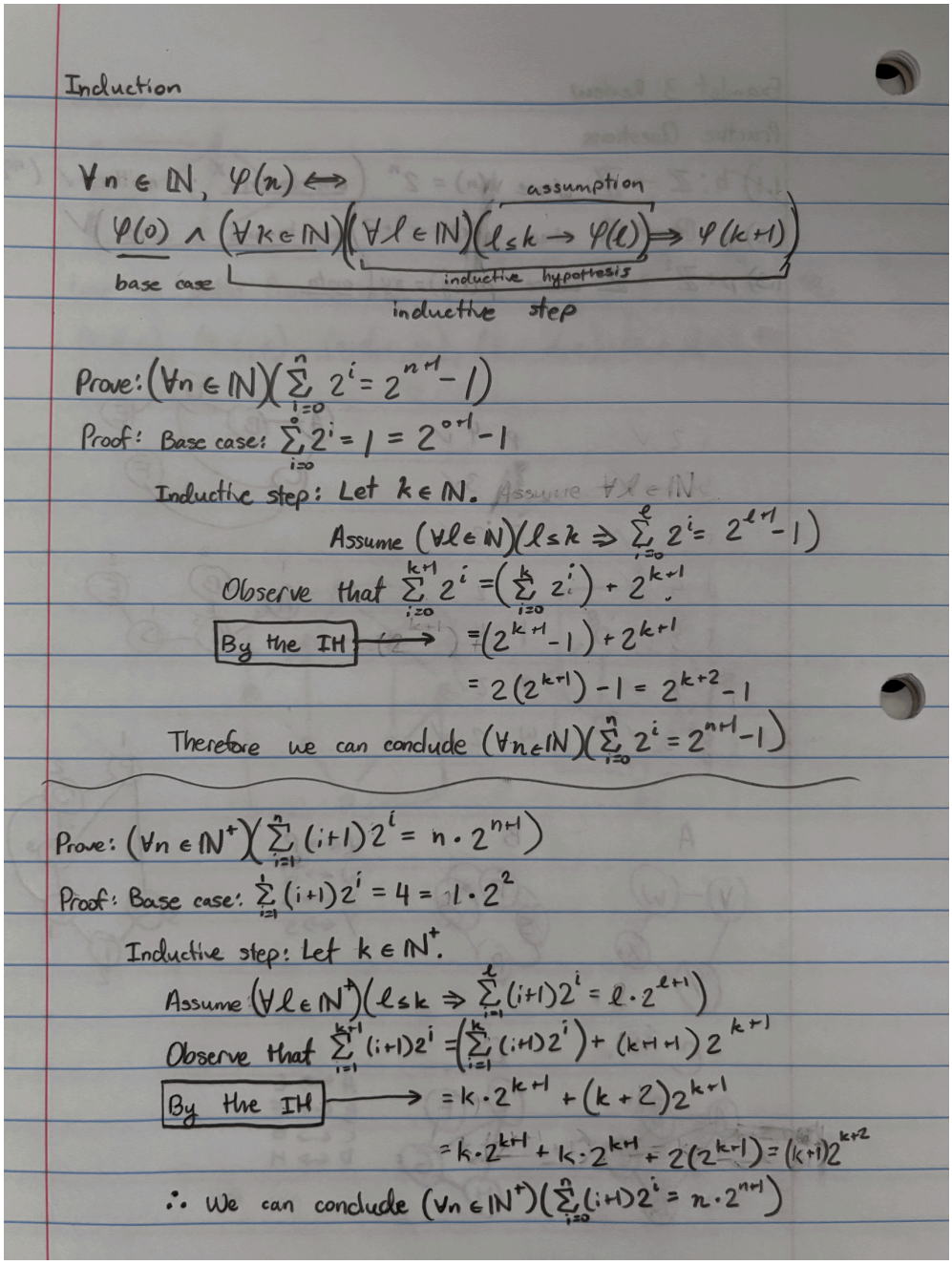
  Because v has at most D neighbors, its neighbors are only using D of the available colors, leaving a spare color that we can assign to v. The coloring of G' can be extended to a coloring of G with $D + 1$ colors.

- This is called a greedy algorithm
  - The performance is very sensitive to the order of the nodes

refer to book for 11.9-11.11 (examples in different contexts)

10.7.25 - Lecture

- P(n) is a predicate
- Examples:



Induction

$\forall n \in \mathbb{N},\ \varphi(n) \iff$

$\varphi(0) \wedge (\forall k \in \mathbb{N})((\forall \ell \in \mathbb{N})(\ell \le k \to \varphi(\ell)) \to \varphi(k+1))$

assumption; inductive hypothesis; base case; inductive step

Prove: $(\forall n \in \mathbb{N})(\sum_{i=0}^{n} 2^i = 2^{n+1} - 1)$

Proof: Base case: $\sum_{i=0}^{0} 2^i = 1 = 2^{0+1} - 1$

Inductive step: Let $k \in \mathbb{N}$. Assume $\forall \ell \in \mathbb{N}$

Assume $(\forall \ell \in \mathbb{N})(\ell \le k \Rightarrow \sum_{i=0}^{\ell} 2^i = 2^{\ell+1} - 1)$

Observe that $\sum_{i=0}^{k+1} 2^i = (\sum_{i=0}^{k} 2^i) + 2^{k+1}$

By the IH $\longrightarrow = (2^{k+1} - 1) + 2^{k+1}$

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

Therefore we can conclude $(\forall n \in \mathbb{N})(\sum_{i=0}^{n} 2^i = 2^{n+1} - 1)$

Prove: $(\forall n \in \mathbb{N}^+)(\sum_{i=1}^{n} (i+1)2^i = n \cdot 2^{n+1})$

Proof: Base case: $\sum_{i=1}^{1} (i+1)2^i = 4 = 1 \cdot 2^2$

Inductive step: Let $k \in \mathbb{N}^+$.

Assume $(\forall \ell \in \mathbb{N}^+)(\ell \le k \Rightarrow \sum_{i=1}^{\ell} (i+1)2^i = \ell \cdot 2^{\ell+1})$

Observe that $\sum_{i=1}^{k+1} (i+1)2^i = (\sum_{i=1}^{k} (i+1)2^i) + (k+1+1)2^{k+1}$

By the IH $\longrightarrow = k \cdot 2^{k+1} + (k+2)2^{k+1}$

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

$\therefore$ We can conclude $(\forall n \in \mathbb{N}^+)(\sum_{i=0}^{n} (i+1)2^i = n \cdot 2^{n+1})$

Prove: $(\forall n \in \mathbb{N})(n > 3 \Rightarrow 2+3n < 2^n)$

Proof: Base case: $2+3(4) < 2^4 \Rightarrow 14 < 16$ ✓

Inductive step: Let $k \in \mathbb{N}$ such that $k > 3$.

Assume $(\forall l \in \mathbb{N})(l > 3 \Rightarrow (l \leq k \Rightarrow 2+3l < 2^l))$

Observe that $2+3(k+1) = (2+3k)+3 < 2^k + 3$ [by IH]

Since $k > 3$, we know $2^k > 2^3 = 8 > 3$.

So $2^k + 3 < 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$

$\therefore 2+3(k+1) < 2^{k+1}$

$\therefore (\forall n \in \mathbb{N})(n > 3 \Rightarrow 2+3n < 2^n)$

Prove: $(\forall n \in \mathbb{N})\left(\sum_{i=0}^{n} \binom{n}{i} = 2^n\right)$

Proof: Base case: $\sum_{i=0}^{0} \binom{0}{0} = 1 = 2^0$

Inductive step: Let $k \in \mathbb{N}$.

Assume $(\forall l \in \mathbb{N})\left(l \leq k \Rightarrow \sum_{i=0}^{l} \binom{l}{i} = 2^l\right)$

Observe that $\sum_{i=0}^{k+1} \binom{k+1}{i} = \left(\sum_{i=0}^{k} \binom{k+1}{i}\right) + \binom{k+1}{k+1}$

$= \sum_{i=1}^{k} \binom{k+1}{i+1} + \binom{k+1}{0} + \binom{k+1}{k+1}$

$= \sum_{i=0}^{k-1} \binom{k+1}{i+1} + \binom{k+1}{0} + \binom{k+1}{k+1}$

$= \sum_{i=0}^{k+1} \binom{k}{i} + \sum_{i=0}^{k-1} \binom{k}{i+1} + \underbrace{\binom{k+1}{0} + \binom{k+1}{k+1}}_{k}$

$= \sum_{i=0}^{k} \binom{k}{i} + \sum_{i=1}^{k} \binom{k}{i} + \binom{k}{0}$

By the IH $\longrightarrow$ $= \sum_{i=0}^{k} \binom{k}{i} + \sum_{i=0}^{k} \binom{k}{i} = 2^k + 2^k$

$= 2^{k+1}$

$\therefore$ We can conclude $(\forall n \in \mathbb{N})\left(\sum_{i=0}^{n} \binom{n}{i} = 2^n\right)$

Week 8: Recursion

12.1 Recursive definitions

- Defines an object in terms of smaller objects of the same type
- Recursive definitions always have two parts:
  - Base case(s)
  - Recursive formula
- Example: $\sum_{i=1}^{n} i$ can be defined as
  - $g(1) = 1$
  - $g(n) = g(n-1) + n$, for all $n \geq 2$

12.2 Finding closed forms

- The simplest technique for finding closed forms is called "unrolling"
  - Substitute a recursive definition into itself (and go back one term)
- Example for $T(1) = 1$ and $T(n) = 2T(n-1) + 3,\ \forall n \geq 2$

$$\begin{aligned} T(n) &= 2T(n-1)+3 \\ &= 2(2T(n-2)+3)+3 \\ &= 2(2(2T(n-3)+3)+3)+3 \\ &= 2^3T(n-3)+2^2\cdot 3+2\cdot 3+3 \\ &= 2^4T(n-4)+2^3\cdot 3+2^2\cdot 3+2\cdot 3+3 \\ &\dots \\ &= 2^kT(n-k)+2^{k-1}\cdot 3+\dots+2^2\cdot 3+2\cdot 3+3 \end{aligned}$$

  - Can be represented as $T(n) = 2^{n+1} - 3$

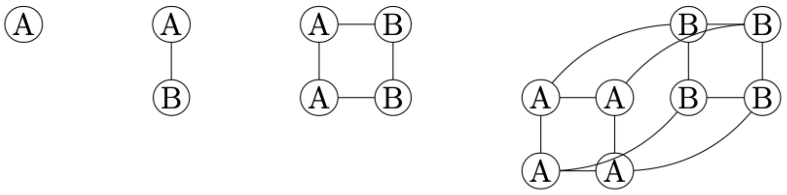
12.3 Divide and conquer

- Standard definition:
  - $S(1) = c$
  - $S(n) = aS(\lceil n/b \rceil) + f(n),\ \forall n \geq 2$

12.4 Hypercubes

- Non-numerical objects can be defined recursively
- Hypercube definition:
  - Q_0 is a single node with no edges
  - Q_n consists of two copies of Q_{n-1} with edges joining corresponding nodes, for any $n \geq 1$

- Hypercube example:



- Hypercubes are a binary coordinate system
- Q_n has 2^n nodes; to compute edges, we can use a recursive definition
  - $E(0) = 0$
  - $E(n) = 2E(n-1) + 2^{n-1}, \forall n \geq 1$

12.5 Proofs with recursive definitions

- Claim: For any $n \geq 0$, F_{3n} is even
- Proof: By induction on n
- Base: $F_0 = 0$, which is even
- Inductive step: $F_{3(k+1)} = F_{3k+3} = F_{3k+2} + F_{3k+1}$.
  Substituting $F_{3k+2} = F_{3k+1} + F_{3k}$ in the equation gives
  $F_{3k+3} = F_{3k+1} + F_{3k} + F_{3k+1} = 2F_{3k+1} + F_{3k}$. By the IH, F_{3k} is even. $2F_{3k+1}$ is even because it is 2 time an integer; so their sum must be even. So $F_{3(k+1)}$ is even, which is what we needed to show.

12.6 Inductive definition and strong induction

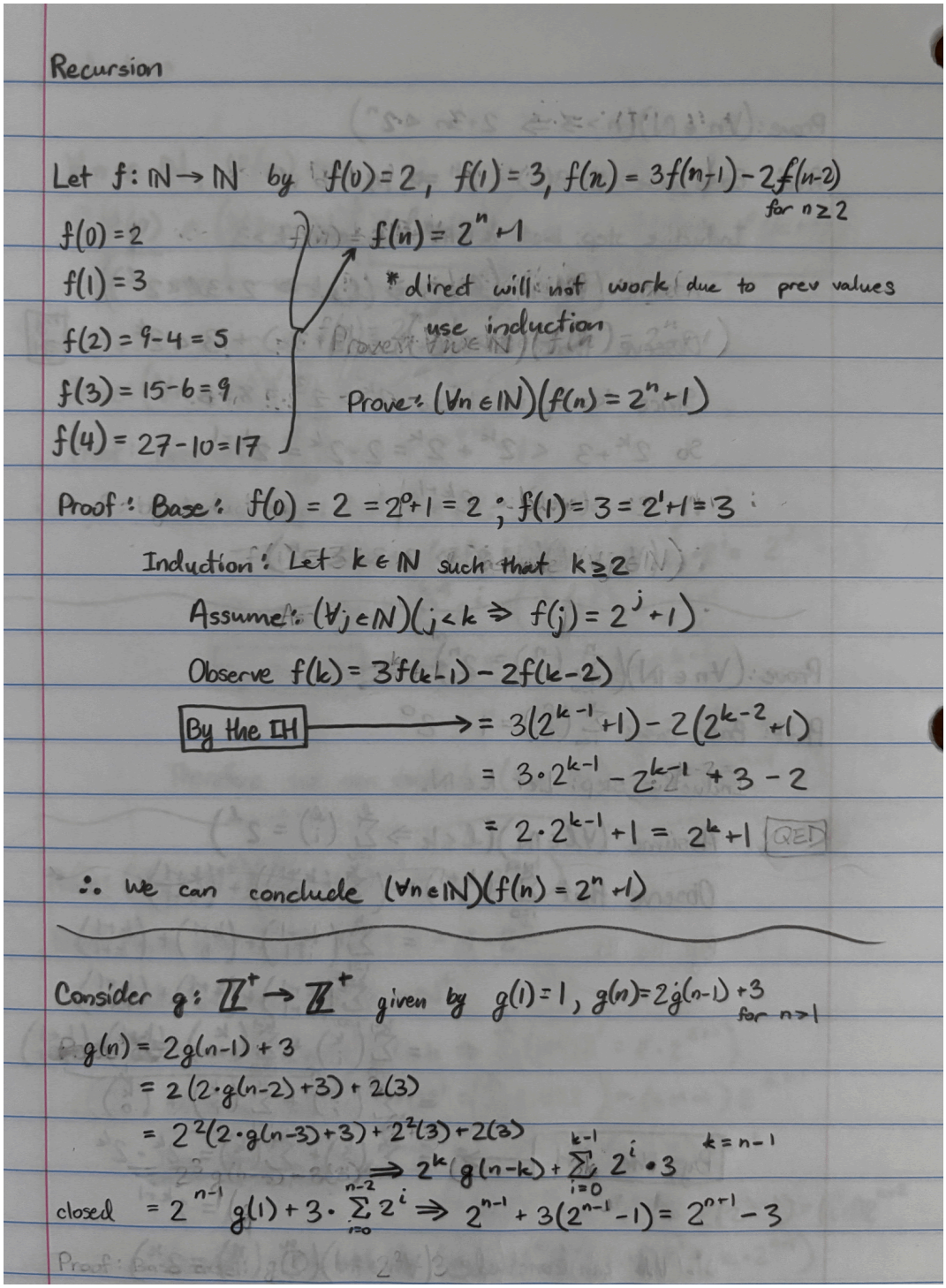
- f is defined as $f(0) = 2, f(1) = 3, f(n+1) = 3f(n) - 2f(n-1), \forall n \geq 1$
- Claim: $\forall n \in \mathbb{N}, f(n) = 2^n + 1$
- Proof: By induction on n
- Base:
  - $f(0)$ is defined to be 2. $2^0 + 1 = 2$. So $f(n) = 2^n + 1$ when $n = 0$
  - $f(1)$ is defined to be 3. $2^1 + 1 = 3$. So $f(n) = 2^n + 1$ when $n = 1$
- Inductive Step: Suppose $f(n) = 2^n + 1$ for $n = 0, 1, \ldots, k$.
  $f(k+1) = 3f(k) - 2f(k-1)$. By the IH, $f(k) = 2^k + 1$ and
  $f(k-1) = 2^{k-1} + 1$. Substituting this, we get
  $f(k+1) = 3(2^k + 1) - 2(2^{k-1} + 1) = 3 \bullet 2^k + 3 - 2^k - 2 = 2 \bullet 2^k + 1$,
  which evaluates to $2^{k+1} + 1$, which is what we needed to show

12.7 Variation in notation

- Recursive definitions are sometimes called inductive definitions or recurrence relations
- The initial condition refers to the base case

10.14.25 - Lecture

- Closed form - direct way to compute a value



Recursion

Let $f: \mathbb{N} \to \mathbb{N}$ by $f(0)=2$, $f(1)=3$, $f(n)=3f(n-1)-2f(n-2)$ for $n \geq 2$

$f(0)=2$

$f(1)=3$

$f(2)=9-4=5$

$f(3)=15-6=9$

$f(4)=27-10=17$

$f(n)=2^n+1$

* direct will not work due to prev values

use induction

Prove: $(\forall n \in \mathbb{N})(f(n)=2^n+1)$

Proof: Base: $f(0)=2=2^0+1=2$; $f(1)=3=2^1+1=3$

Induction: Let $k \in \mathbb{N}$ such that $k \geq 2$

Assume: $(\forall j \in \mathbb{N})(j<k \Rightarrow f(j)=2^j+1)$

Observe $f(k)=3f(k-1)-2f(k-2)$

By the IH $\longrightarrow = 3(2^{k-1}+1)-2(2^{k-2}+1)$

$= 3 \cdot 2^{k-1} - 2^{k-1} + 3 - 2$

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

$\therefore$ we can conclude $(\forall n \in \mathbb{N})(f(n)=2^n+1)$

Consider $g: \mathbb{Z}^+ \to \mathbb{Z}^+$ given by $g(1)=1$, $g(n)=2g(n-1)+3$ for $n>1$

$g(n)=2g(n-1)+3$

$= 2(2 \cdot g(n-2)+3)+2(3)$

$= 2^2(2 \cdot g(n-3)+3)+2^2(3)+2(3)$

$\Rightarrow 2^k g(n-k) + \sum_{i=0}^{k-1} 2^i \cdot 3$ $\quad k=n-1$

closed $= 2^{n-1} g(1) + 3 \cdot \sum_{i=0}^{n-2} 2^i \Rightarrow 2^{n-1} + 3(2^{n-1}-1) = 2^{n+1}-3$

Proof:

Proof: Base case: $g(1) = 1 = 2^2 - 3$

Induction: Let $k \in \mathbb{N}^+$ such that $k > 1$

Assume $(\forall j \in \mathbb{N}^+)(j < k \Rightarrow g(j) = 2^{j+1} - 3)$

Observe $g(k) = 2g(k-1) + 3$

By the IH $\longrightarrow = 2(2^{k-1+1} - 3) + 3$

$= 2 \cdot 2^k - 2(3) + 3$

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

$\therefore$ We can conclude $(\forall n \in \mathbb{N}^+)(g(n) = 2^{n+1} - 3)$

Prove $(\forall n \in \mathbb{N}^+)(T(n) = 4\log_2(n) + 4)$ $\quad T(1) = 4$, $T(n) = T(\frac{n}{2}) + 4$ $\quad$ BINARY SEARCH

Proof: Base case: $T(1) = 4 = 0 + 4 = 4\log_2(1) + 4$

Induction: Let $k \in \mathbb{N}^+$

Assume: $(\forall j \in \mathbb{N})(j < k \Rightarrow T(j) = 4\log_2(j) + 4)$

Observe $T(k) = T(\frac{k}{2}) + 4$

By the IH $\longrightarrow = (4\log_2(\frac{k}{2}) + 4) + 4$

$= 4(\log_2 k - \log_2(2)) + 4 + 4$

$= 4(\log_2 k - 1) + 4 + 4$

$= 4\log_2 k + 4$

$\therefore$ We can conclude $(\forall n \in \mathbb{N}^+)(T(n) = 4\log_2(n) + 4)$

$$\begin{cases} T(1) := 4 \\ T(n) := T(n/2) + 4 \end{cases} \qquad T : \mathbb{N}_+ \to \mathbb{N}$$

$T(n) = T(\frac{n}{2}) + 4 = T(\frac{n}{2^1}) + \boxed{2^1} \cdot 4$

$= \left(T(\frac{n}{4}) + 4\right) + 4 = T(\frac{n}{2^2}) + \boxed{2^2} 4$

$= \left(T(\frac{n}{2^3}) + 4\right) + 2 \cdot 4 = T(\frac{n}{2^3}) + \boxed{3} \cdot 4$

$\vdots$

$= T(\frac{n}{2^k}) + k \cdot 4 \qquad \frac{n}{2^k} = 1 \iff n = 2^k \iff k = \log_2(n)$

$\vdots$

$= T(\frac{n}{2^{\log_2(n)}}) + \log_2(n) \cdot 4$

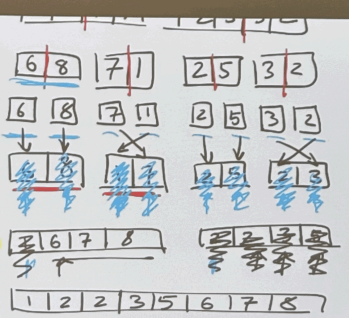
$= T(1) + \log_2(n) \cdot 4$

$= \boxed{4\log_2(n) + 4}$

$T: \mathbb{N}_+ \to \mathbb{N}$

$T(1) := 0$

$T(n) := T(\frac{n}{2}) + T(\frac{n}{2}) + \frac{n}{2} + \frac{n}{2} = 2T(\frac{n}{2}) + n$



$$
\begin{aligned}
T(n) &= 2T(\tfrac{n}{2}) + n \\
&= 2\left(2T(\tfrac{n}{4}) + \tfrac{n}{2}\right) + n = 4 \cdot T(\tfrac{n}{4}) + 2n \\
&= 4\left(2T(\tfrac{n}{8}) + \tfrac{n}{4}\right) + 2n = 8T(\tfrac{n}{8}) + 3n \\
&\vdots \\
&= 2^k T(\tfrac{n}{2^k}) + kn \qquad \tfrac{n}{2^k} = 1 \iff n = 2^k \iff k = \log_2(n) \\
&\vdots \\
&= 2^{\log_2(n)} T\left(\tfrac{n}{2^{\log_2(n)}}\right) + \log_2(n) \cdot n \\
&= n \cdot 0 + \log_2(n) \cdot n \\
&= n \cdot \log_2(n).
\end{aligned}
$$

Prove $(\forall n \in \mathbb{N}^+)(T(n) = n \log_2(n))$ $\quad T(1)=0,\ T(n) = 2(T(n/2)) + n$ MERGE SORT

Proof: Base case: $T(1) = 0 = 1 \cdot \log_2(1)$

Induction: Let $k \in \mathbb{N}^+$ such that $k > 1$

Assume $(\forall j \in \mathbb{N}^+)(j < k \Rightarrow T(j) = j \log_2(j))$

Observe $T(k) = 2(T(\frac{k}{2})) + k$

By the IH $\longrightarrow = 2\left(\frac{k}{2}\log_2(\frac{k}{2})\right) + k$

$= 2\left(\frac{k}{2}\log_2(k) - \frac{k}{2}\log_2(2)\right) + k$

$= k\log_2 k - k + k = k\log_2(k)$

$\therefore$ We can conclude $(\forall n \in \mathbb{N})(T(n) = n\log_2(n))$

Week 9: Trees and Grammars

13.2 Defining Trees

- A tree is an undirected graph that has a special node called the root, where every node is connected to it with exactly one path
    - A leaf node is a node with no children
- Nodes of a tree are organized in levels, where the root is level 0, and its children are level 1
    - The height of a tree is the maximum length of a path from the root to a leaf

13,3 m-ary trees

- An m-ary tree allows each node to have up to m children
    - In a FULL m-ary tree, each node either has 0 children or m children
    - In a COMPLETE m-ary tree, all leaves are at the same height
    - In a COMPLETE AND FULL m-ary tree, the whole bottom level is populated with leaves

13.4 Height vs number of nodes

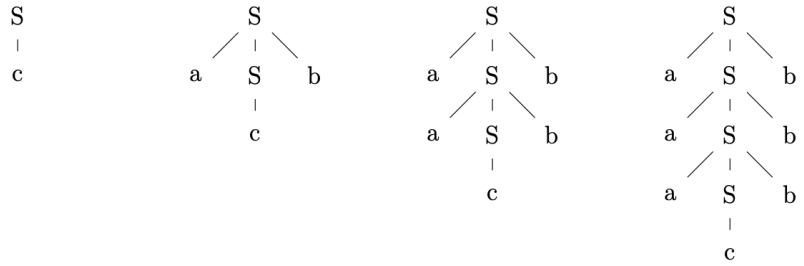
- The total number of nodes in a full and complete binary tree is $2^{h+1} - 1$, which is proportional to $log_2 n$
- Balanced binary trees are binary trees where all leaves are approximately the same height

13.5 Context-free grammars

- Terminal symbols can be words or characters
- Non-leaf nodes contain symbols that help indicate the structure of the sentence
- A context-free grammar is a set of rules that specify what sorts of children are possible for a parent node with each type of label
    - If all the nodes of a tree have children matching the rules of some grammar G, the tree T and the terminal sequence are generated by G
- Two details that specify which trees are allowed by a grammar
    - Give a set of terminals (symbols that are allowed to appear on leaf nodes)

- State start symbols (symbols that are allowed to appear on the root node)

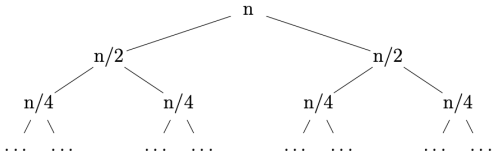
- Ex: $S \to aSb$, $S \to c$



- Sometimes it is convenient to have a branch of a parse tree produce nothing, which is done by having a rule where the right-hand side is ϵ (empty string)

13.6 Recursion trees

- Ex: $S(1) = c, S(n) = 2S(n/2) + n, n \geq 2$



To sum everything in the tree, we need to ask:

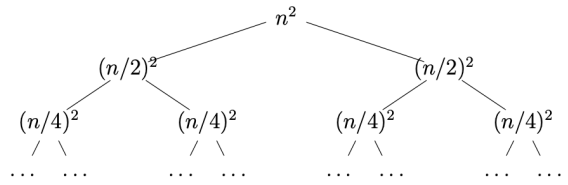
- How high is this tree, i.e. how many levels do we need to expand before we hit the base case $n = 1$?
- For each level of the tree, what is the sum of the values in all nodes at that level?
- How many leaf nodes are there?

- In this example, the tree has a height of $log(n)$, the sum of all non-leaf nodes is $n\,log(n)$, and n leaf nodes contribute c to the sum. The result is $n\,log(n) + cn$

13.7 Another recursion tree example

$$\begin{aligned} P(1) &= c \\ P(n) &= 2P(n/2) + n^2, \qquad \forall n \geq 2 \ \ (n \text{ a power of } 2) \end{aligned}$$

Its recursion tree is



- In this example, the tree has a height of $log(n)$, the sum of all nodes at level k is $\frac{n^2}{2^k}$.

$$\begin{aligned} P(n) &= \sum_{k=0}^{\log n - 1} n^2 \frac{1}{2^k} = n^2 \sum_{k=0}^{\log n - 1} \frac{1}{2^k} \\ &= n^2(2 - \frac{1}{2^{\log n - 1}}) = n^2(2 - \frac{2}{2^{\log n}}) = n^2(2 - \frac{2}{n}) = 2n^2 - 2n \end{aligned}$$

- Adding cn to cover the leaf nodes, our final closed form is $2n^2 + (c-2)n$.

13.8 Tree induction

- View a tree as consisting of a root node plus some number of subtrees rooted at its children
- Claim: Let T be a binary tree, with height h and n nodes. Then $n \leq 2^{h+1} - 1$

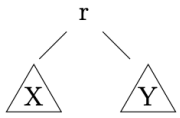
Proof by induction on h, where h is the height of the tree.

Base: The base case is a tree consisting of a single node with no edges. It has $h = 0$ and $n = 1$. Then we work out that $2^{h+1} - 1 = 2^1 - 1 = 1 = n$.

Induction: Suppose that the claim is true for all binary trees of height $< h$. Let T be a binary tree of height h ($h > 0$).

Case 1: T consists of a root plus one subtree X. X has height $h - 1$. So X contains at most $2^h - 1$ nodes. T only contains one more node (its root), so this means T contains at most 2^h nodes, which is less than $2^{h+1} - 1$.

Case 2: T consists of a root plus two subtrees X and Y. X and Y have heights p and q, both of which have to be less than h, i.e. $\leq h - 1$. X contains at most $2^{p+1} - 1$ nodes and Y contains at most $2^{q+1} - 1$ nodes, by the inductive hypothesis. But, since p and q are less than h, this means that X and Y each contain $\leq 2^h - 1$ nodes.



So the total number of nodes in T is the number of nodes in X plus the number of nodes in Y plus one (the new root node). This is $\leq 1 + (2^p - 1) + (2^q - 1) \leq 1 + 2(2^h - 1) = 1 + 2^{h+1} - 2 = 2^{h+1} - 1$

So the total number of nodes in T is $\leq 2^{h+1} - 1$, which is what we needed to show. $\square$

13.9 Heap example

- Heap property - a tree structure where the parent node is at least as large as each of its children
    - The path from leaf to root is always increasing

- Claim: If a full binary tree has the heap property, then the value in the root of the tree is at least as large as the value in any other node in the tree
- Proof: by induction on the tree height h

  Base: $h = 0$. A tree of height zero contains only one node, so obviously the largest value in the tree lives in the root!

  Induction: Suppose that the claim is true for all full binary trees of height $< h$. Let T be a tree of height h ($h > 0$) which has the heap property. Since T is a full binary tree, its root r has two children p and q. Suppose that X is the subtree rooted at p and Y is the subtree rooted at q.

  Both X and Y have height $< h$. Moreover, notice that X and Y must have the heap property, because they are subtrees of T and the heap property is a purely local constraint on node values.

  Suppose that x is any node of T. We need to show that $v(r) \geq v(x)$. There are three cases:

  Case 1: $x = r$. This is obvious.

  Case 2: x is any node in the subtree X. Since X has the heap property and height $\leq h$, $v(p) \geq v(x)$ by the inductive hypothesis. But we know that $v(r) \geq v(p)$ because T has the heap property. So $v(r) \geq v(x)$.

  Case 3: x is any node in the subtree Y. Similar to case 2.

  So, for any node x in T, $v(r) \geq v(x)$.$\square$

13.10 Proof using grammar trees

- Claim: all trees generated by G have the same number of nodes with label a as with label b; $S \to ab,\ S \to SS,\ S \to aSb$

Proof by induction on the tree height h.

Base: Notice that trees from this grammar always have height at least 1. The only way to produce a tree of height 1 is from the first rule, which generates exactly one a node and one b node.

Induction: Suppose that the claim is true for all trees of height $< k$, where $k \geq 1$. Consider a tree T of height k. The root must be labelled S and the grammar rules give us three possibilities for what the root's children look like:

Case 1: The root's children are labelled a and b. This is just the base case.

Case 2: The root's children are both labelled S. The subtrees rooted at these children have height $< k$. So, by the inductive hypothesis, each subtree has equal numbers of a and b nodes. Say that the left tree has m of each type and the right tree has n of each type. Then the whole tree has $m + n$ nodes of each type.

Case 3: The root has three children, labelled a, S, and b. Since the subtree rooted at the middle child has height $< k$, it has equal numbers of a and b nodes by the inductive hypothesis. Suppose it has m nodes of each type. Then the whole tree has $m+1$ nodes of each type.

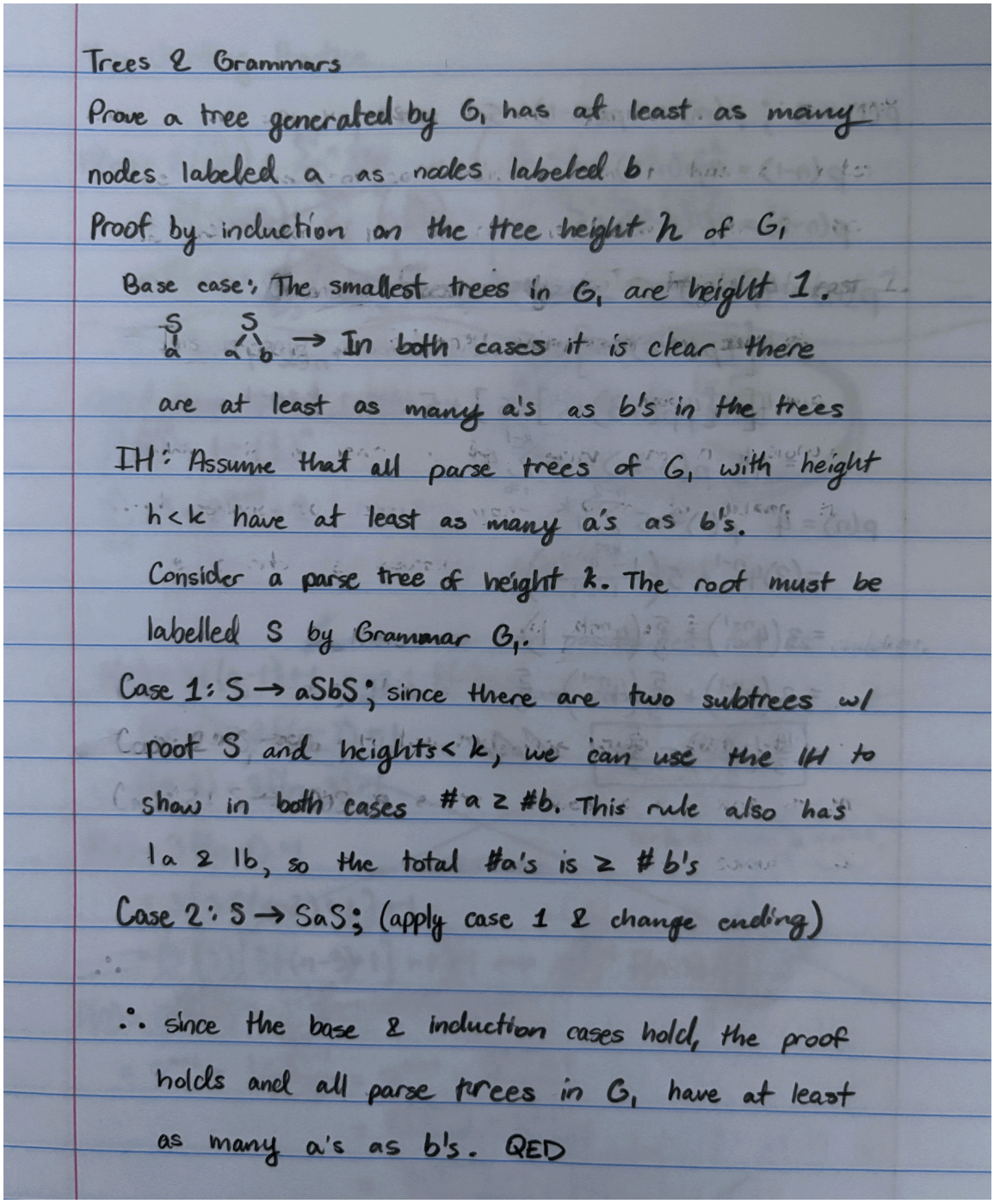
In all three cases, the whole tree T has equal numbers of a and b nodes.

10.21.25 - Lecture

Grammar Trees

Define a grammar G_1 by $S \to aSbS \mid SaS \mid ab \mid a$. where S is the only start symbol and the terminal symbols are a and b. Prove that a tree generated by G_1 has at least as many nodes labeled a as nodes labeled b.

Solution:

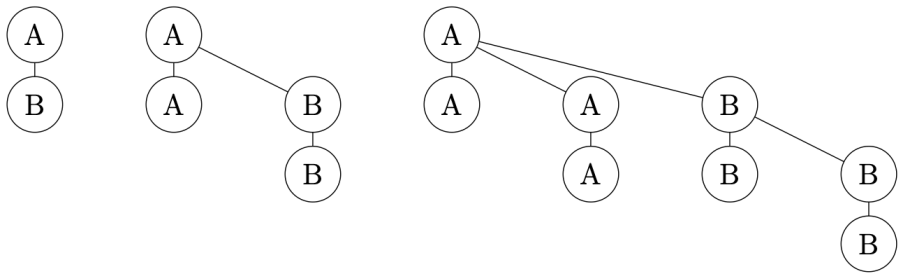
- 

A **m-nomial** of order m is defined recursively as follows:

(1) A single root node is a m-nomial tree of order 0.

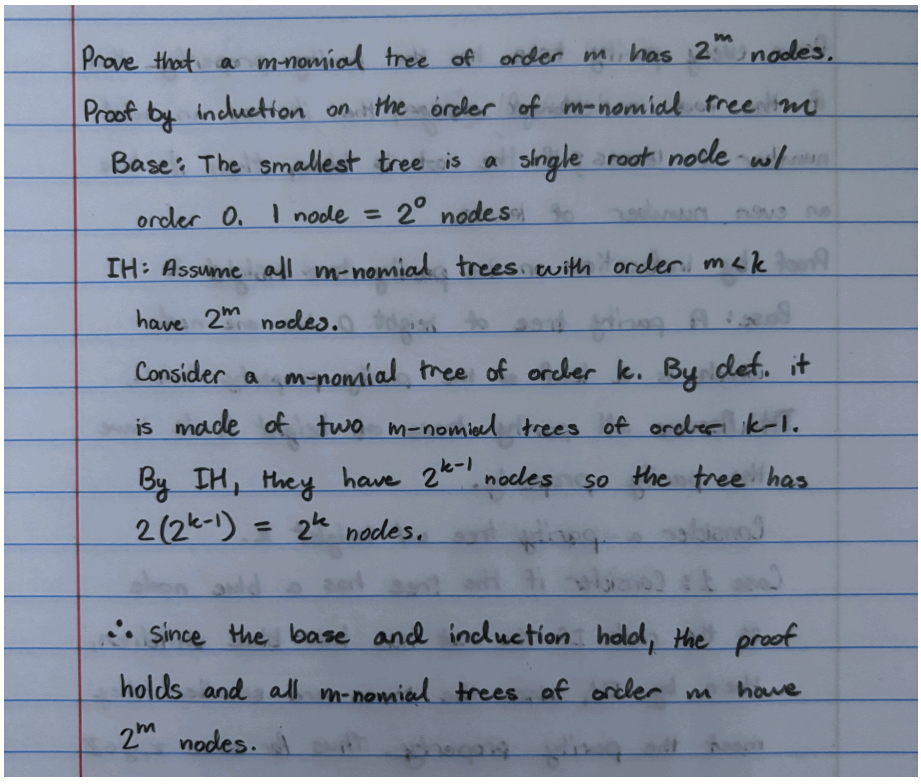
(2) A m-nomial tree of order m consists of two m-nomial trees of order $m-1$, with the root of the second connected as the rightmost child of the root of the first.

The following picture shows the m-nomial trees of order 1, 2, and 3. The labels on the nodes show how the larger tree is divided into two lower-order subtrees.



Use induction on the order of the tree to prove that a m-nomial tree of order m has 2^m nodes.

Solution:

- 

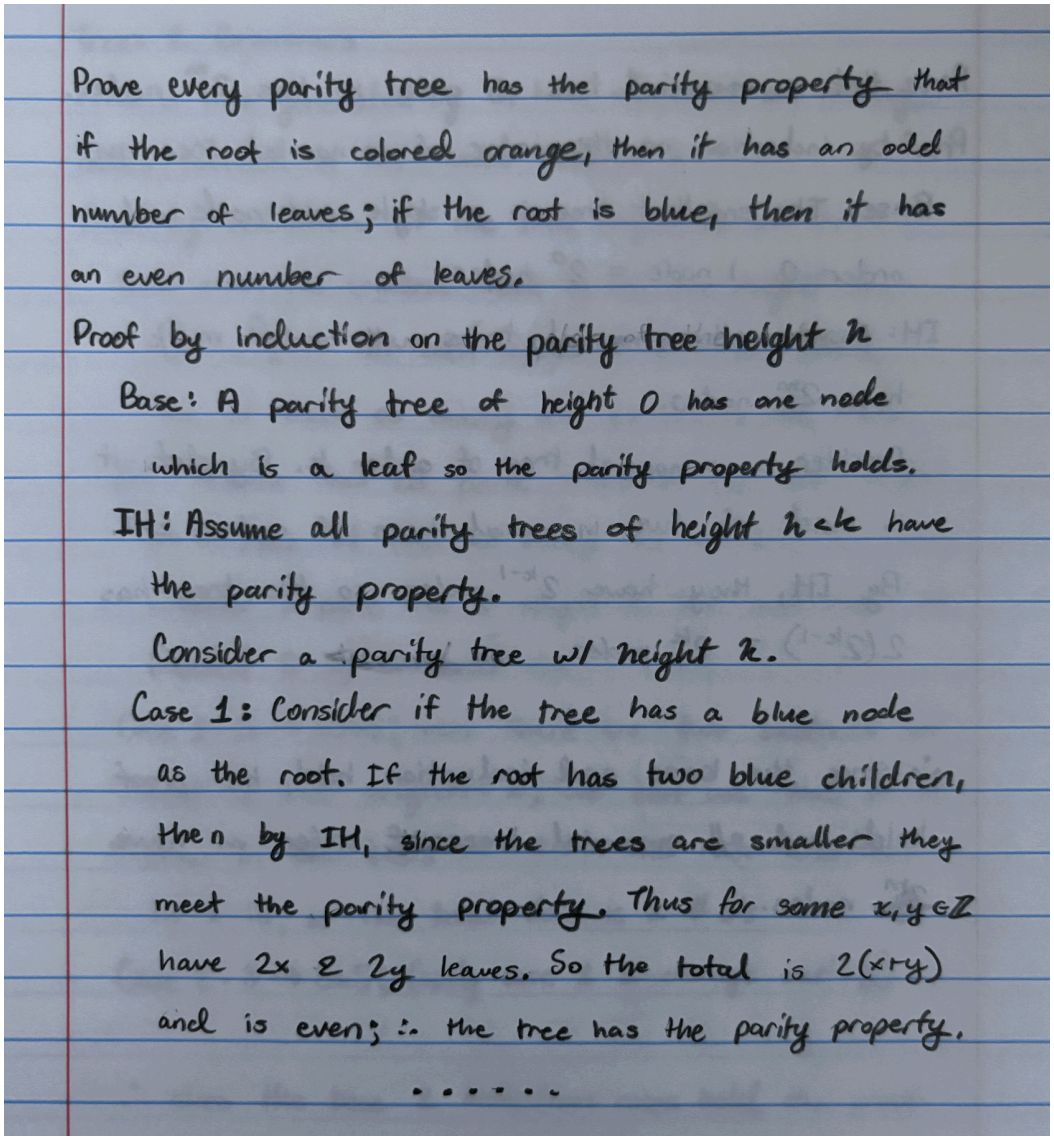
Parity Trees

A parity tree is a full binary tree with each node colored orange or blue such that:

1. If v is a leaf node, then v is colored orange.
2. If v has two children of the same color, then v is colored blue.
3. If v has two children of different colors, then v is colored orange.

Prove by induction that every parity tree has the parity property that if the root is colored orange, then it has an odd number of leaves; and if the root is colored blue, then it has an even number of leaves.

Solution:

- 

Week 10: Big-O and Algorithms

14.1 Running times of programs

- Ignore multiplicative constants
- Ignore behavior on small inputs (only analyze large ones)

14.2 Asymptotic relationships

- When a function is the sum of faster and lower growing terms, only focus on the faster growing term
- Asymptotically similar: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$
- Asymptotically smaller: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

14.3 Ordering primitive functions

- Higher-order polynomials grow faster than lower-order polynomials
- Exponentials grow faster than polynomials
- $2^n << n!$
- $1 << log(n) << n << nlog(n) << n^2$

14.4 The dominant term method

- Dominant term - term that grows the fastest

14.5 Big-O

> There are positive real numbers c and k such that $0 \leq f(n) \leq cg(n)$ for every $n \geq k$.

> The factor c in the equation models the fact that we don't care about multiplicative constants, even on the dominant term. The function $f(n)$ is allowed to wiggle around as much as it likes, compared to $cg(n)$, as long as it remains smaller. And we're entirely ignoring what f does on inputs smaller than k.

- If $f(n)$ is asymptotically smaller than $g(n)$, then $f(n) = O(g(n))$
- When $f(n) = O(g(n)$ and $g(n) = O(f(n)$, then $f(n) = \Theta(g(n))$

14.6 Applying the definition of Big-O

- Overkill makes it easier to confirm if chosen values work correctly

14.7 Proving a primitive function relationship

- Claim: For every positive integer $n \geq 4$, $\frac{2^n}{n!} < \left(\frac{1}{2}\right)^{n-4}$

Proof: Suppose that n is an integer and $n \geq 4$. We'll prove that $\frac{2^n}{n!} < (\frac{1}{2})^{n-4}$ using induction on n.

Base: $n = 4$. [show that the formula works for $n = 4$]

Induction: Suppose that $\frac{2^n}{n!} < (\frac{1}{2})^{n-4}$ holds for $n = 4, 5, \ldots, k$. And, in particular, $\frac{2^k}{k!} < (\frac{1}{2})^{k-4}$

We need to show that the claim holds for $n = k + 1$, i.e. that $\frac{2^{k+1}}{(k+1)!} < (\frac{1}{2})^{k-3}$

By our inductive hypothesis $\frac{2^k}{k!} < (\frac{1}{2})^{k-4}$. So $\frac{1}{2} \cdot \frac{2^k}{k!} < (\frac{1}{2})^{k-3}$.

So then we have $\frac{2^{k+1}}{(k+1)!} < \frac{1}{2} \cdot \frac{2^k}{k!} < (\frac{1}{2})^{k-3}$ which is what we needed to show.

15.2 Basic data structures

- An array provides constant time O(1) access to any element; however, changing the array length takes O(n) time
- In a linked list, each object points to the next object in the list
- It takes O(1) time to add, remove, or read/write the value at the head of the list, along with locations that are a constant number of places from the head (i.e. the tenth element of a list)
    - Accessing slightly earlier values or removing values from the tail has O(n) time

15.3 Nested loops

- Each loop adds O(n) runtime
    - Two nested for loops have a runtime of $O(n^2)$

15.4 A connected component algorithm

```
01  component(G: a graph; s: node in G)
02        Unmark all nodes of G.
03        RV = emptylist
04        Q = emptylist
05        Mark node s and add it to Q
06        while (Q is not empty)
07              p = first(Q)
08              Q = rest(Q)
09              add p to RV
10              for every node n that is a neighbor of p in G
11                    if n is not marked
12                          mark n
13                          add n to the tail of Q
14        return RV
```

- Figure 15.2: Return all nodes connected to s

- The algorithm runs through the list twice, so the runtime is $O(n^2)$
- Worst-case analysis - longest runtime in an algorithm

15.5 Binary search

```
01  squareroot(n: positive integer)
02          p = squarerootrec(n, 1, n)
03          if (n − p^2 ≤ (p + 1)^2 − n)
05                  return p
05          else return p + 1


11  squarerootrec(n, bottom, top: positive integers)
12          if (bottom = top) return bottom
13          middle = floor((bottom+top)/2)
14          if (middle^2 == n)
15                  return middle
16          else if (middle^2 ≤ n)
17                  return squarerootrec(n,middle,top)
18          else
19                  return squarerootrec(n,bottom,middle)
```

Figure 15.3: Binary search for $\sqrt{n}$

- This can be represented as a recursive definition where $T(1) = c,\ T(n) = T(n/2) + d$
- Unrolled k times ($T(n) = T(n/2^k) + kd$), we can figure out the runtime is $O(log\ n)$

15.6 Merging two lists

```
01  merge(L1,L2: sorted lists of real numbers)
02          if (L1 is empty)
03                  return L2
04          else if (L2 is empty)
05                  return L1
06          else if (head(L1) <= head(L2))
07                  return cons(head(L1),merge(rest(L1),L2))
08          else
09                  return cons(head(L2),merge(L1,rest(L2)))
```

Figure 15.4: Merging two lists

- For merging, the size is the sum of the two lengths
  - $T(1) = c,\ T(n) = T(n-1) + d$
  - So, the runtime is $O(n)$

15.7 Mergesort

```
01  mergesort(L = a_1, a_2, ..., a_n: list of real numbers)
02          if (n = 1) then return L
03          else
04                  m = ⌊n/2⌋
05                  L_1 = (a_1, a_2, ..., a_m)
06                  L_2 = (a_{m+1}, a_{m+2}, ..., a_n)
07                  return merge(mergesort(L_1),mergesort(L_2))
```

Figure 15.5: Sorting a list using mergesort

-
- Mergesort makes two recursive calls to itself
  - $T(1) = c,\ T(n) = 2T(n/2) + dn$
  - The tree has $O(log\ n)$ non-leaf levels, and the work on each level sums up to dn
  - So, the runtime is $O(n\ log\ n)$

15.8 Tower of Hanoi

```
01  hanoi(A,B,C: pegs, d_1, d_2 ... d_n: disks)
02          if (n = 1) move d_1 = d_n from A to B.
03          else
04                  hanoi(A,C,B,d_1, d_2, ... d_{n-1})
05                  move d_n from A to B.
06                  hanoi(C,B,A,d_1, d_2, ... d_{n-1})
```

Figure 15.6: Solving the Towers of Hanoi problem

-
- Recursively, this is $T(1) = c,\ T(n) = 2T(n-1) + d$
  - Unrolling this results with a runtime of $O(2^n)$

15.9 Multiplying big integers

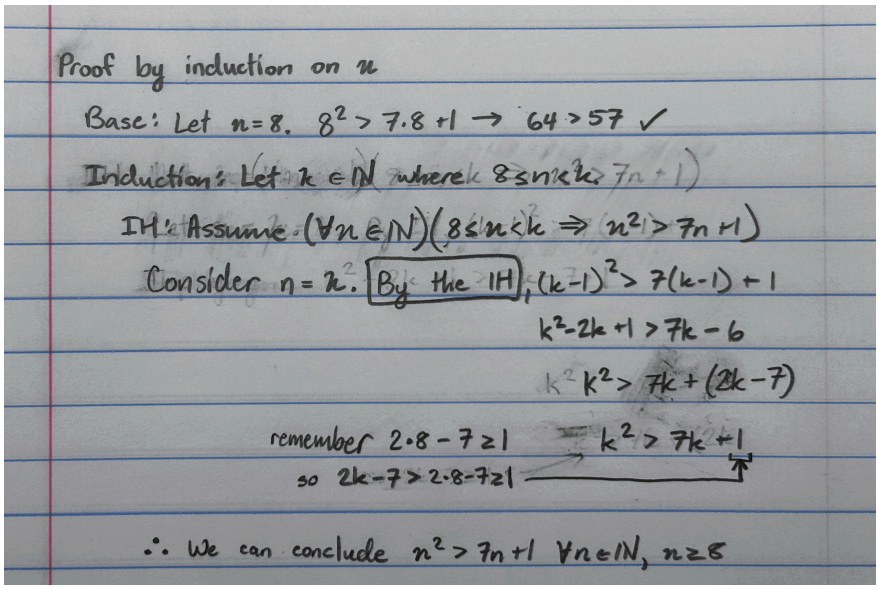
- When multiplying large numbers, normal CPU instructions aren't enough, so the numbers are broken into sequences of digits (often base 2)
  - This recurrence is $T(n) = 4T(n/2) + dn$ and has a runtime of $O(n^2)$
  - The trick to speeding up the algorithm is to rewrite the algebra for computing
    - $B = (x_1 + x_0)(y_1 + y_0) - A - C$
  - Leads to a recursive definition $T(n) = 3T(n/2) + O(n)$

10.28.25 - Lecture

Induction with Inequalities

Prove that $n^2 > 7n + 1$ for all integers $n \geq 8$

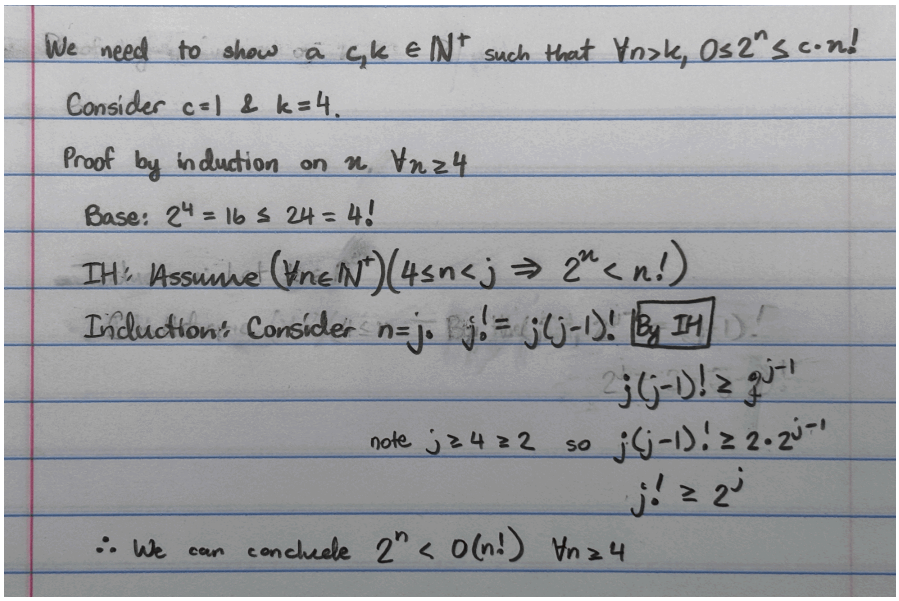
Solution:

- 

Big-O Analysis

Prove that 2^n is $O(n!)$

Solution:

- 

Big-O Analysis

Consider two functions $f(n)$ which is $O(2^n)$ and $g(n)$ which is $O(n!)$. Is it then the case that $f(n)$ is $O(g(n))$?

Solution:

- It is never the case $f(n) = O(g(n))$

Week 11: Algorithms (continued)

- Continuation of Week 10 content

11.4.25 - Lecture

Mystery Code I

In line 05 the procedure MA is calling itself on a version of the input list with the kth element (a_k) removed. Assume it takes constant time to temporarily remove a_k from the list. (Doing this in constant time actually requires some extra details that we are hiding for clarity.)

```
00 MA(a_1, ..., a_n) : list of n positive integers, n ≥ 3)
01        if (n = 3) return a_1 + a_2 + a_3
02        else
03              bestval = 0
04              for k = 1 to n
05                    newval = MA(a_1, a_2, ..., a_{k-1}, a_{k+1}, ... a_n)
06                    if (newval > bestval) bestval = newval
07              end for
08              return bestval
```

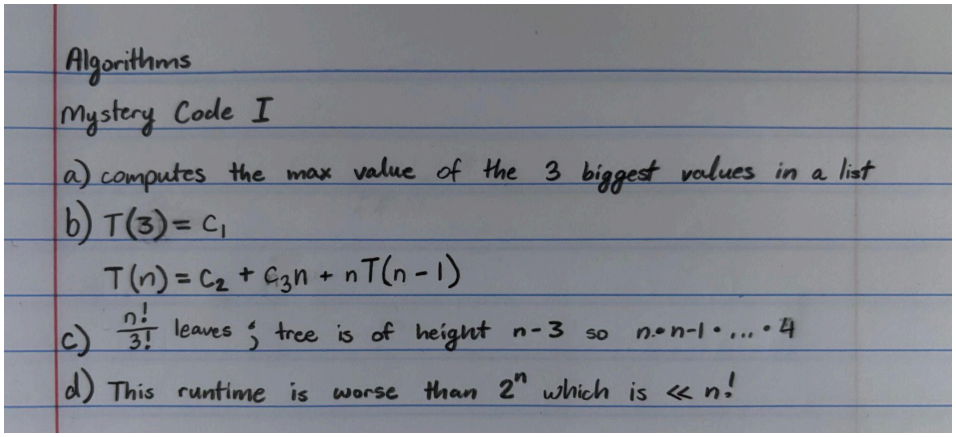
(a) Describe (in English) what MA computes.

(b) Suppose that $T(n)$ is the running time of MA on an input array of length n. Give a recursive definition of $T(n).$

(c) How many leaf nodes are there in the recursion tree for $T(n)$? Briefly explain.

(d) Does MA run in $O(2^n)$ time? Briefly explain why or why not.

Solution:

- 

Mystery Code II

Consider an array of n *distinct* real numbers $a_1, a_2, \ldots, a_n$. We say that the array has a *peak* at position k if the following two conditions hold for every position j between 2 and n:

(1) If $j \leq k$, then $a_{j-1} < a_j$.

(2) If $j > k$, $a_{j-1} > a_j$.

Consider the following procedure to determine position of the peak of an array (assume that the array does indeed have a peak):

```
00  procedure Find(a_1, a_2, ..., a_n: array of real numbers)
01      if (n = 1)
02          return 1
03      if (a_1 > a_2)
04          return 1
05      else if (a_n > a_{n-1})
06          return n
07      k = floor((1+n)/2)
08      if (a_{k-1} > a_k)
09          return Find(a_1, ..., a_{k-1})
10      else if (a_k < a_{k+1})
11          return Find(a_{k+1}, ..., a_n) + k
12      else
13          return k
```

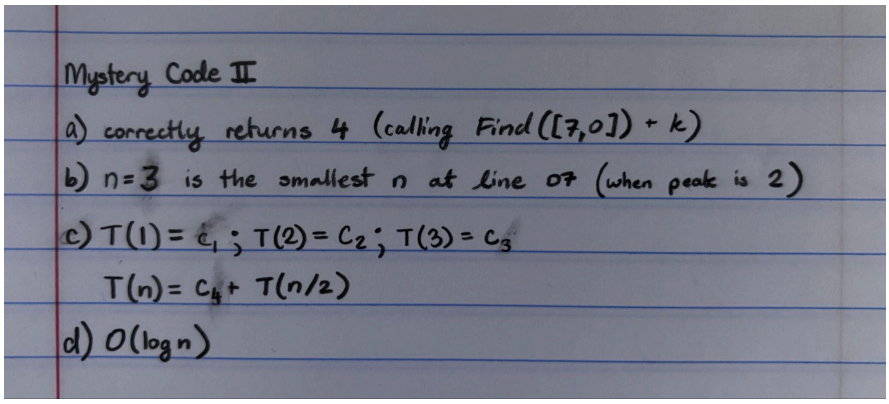
(a) Consider the array $-1, 3, 6, 7, 0$. Trace the execution of the above pseudocode and show that it correctly returns the position of the peak.

(b) At line 07, what is the smallest value that n might contain? Why?

(c) Let $T(n)$ be the worst-case running time of the above pseudocode when the array has size n. Write a recurrence for $T(n)$, including the necessary base case(s). Assume that splitting the array (lines 09 and 11) takes constant time.

(d) What is the tightest big-O running time of Find?

Solution:

- 

Week 12: Contradiction

17.1 The method

- In proof by contradiction, we show a claim *P* is true by showing the negation ¬*P* leads to a contradiction
    - A contradiction is a statement that is well-known to be false
- Claim: There is no largest even integer
  Proof: Suppose not. Suppose that there were a largest even integer and call it k. Since k is even, it has the form $2n$ for some $n \in \mathbb{Z}$. Consider $k + 2$. $k + 2 = 2n + 2 = 2(n + 1)$. So, $k + 2$ is even. But, $k + 2$ is larger than k. This contradicts our assumption that k was the largest even integer, so the original claim must be true.

17.2 √2 is irrational

- Claim: $\sqrt{2}$ is irrational
- Proof: Suppose not. That is, suppose $\sqrt{2}$ is rational. Then we can write $\sqrt{2}$ as a fraction $\frac{a}{b}$ where a and b are integers with no common factors. Since $\sqrt{2} = \frac{a}{b}$, $2 = \frac{a^2}{b^2}$. So, $2b^2 = a^2$. By definition of even, this means a^2 is even, so $a = 2n$ for some integer n.
  If $a = 2n$, then $2b^2 = 4n^2$, so $b^2 = 2n^2$. This means b^2 is even, so b must be even.
  We now have a contradiction because a and b should not have any common factors, yet they are both divisible by 2. Therefore, it must be the case that $\sqrt{2}$ is irrational.

17.3 There are infinitely many prime numbers

- Claim: There are infinitely many prime numbers

  Proof: Suppose not. That is, suppose there were only finitely many prime numbers. Let's call them p_1, p_2, up through p_n.

  Consider $Q = p_1 p_2 \cdots p_n + 1$.

  If you divide Q by one of the primes on our list, you get a remainder of 1. So Q isn't divisible by any of the primes p_1, p_2, up through p_n. However, by the Fundamental Theorem of Arithmetic, Q must have a prime factor (which might be either itself or some smaller number). This contradicts our assumption that $p_1, p_2, \ldots p_n$ was a list of all the prime numbers. □

17.4 Lossless compression

Claim 52 *A lossless compression algorithm that makes some files smaller must make some (other) files larger.*

Proof: Suppose not. That is, suppose that we had a lossless compression algorithm A that makes some files smaller and does not make any files larger.

Let x be the shortest file whose compressed size is smaller than its original size. (If there are two such files of the same length, pick either at random.) Suppose that the input size of x is m characters.

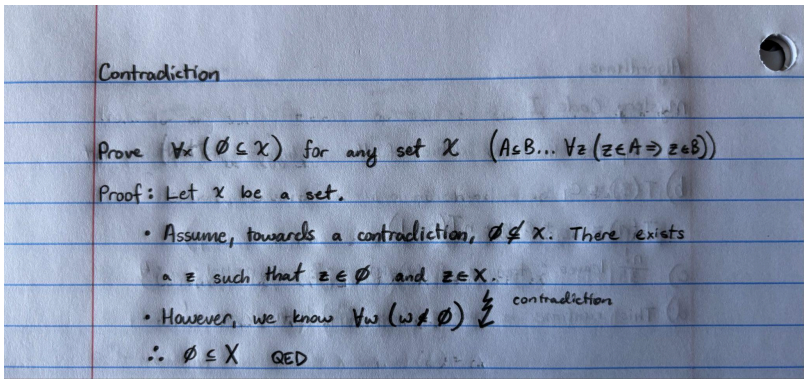
Suppose that S is the set of distinct files with fewer than m characters. Because x shrinks, A compresses x to a file in S. Because no files smaller than x shrink, each file in S compresses to a file (perhaps the same, perhaps different) in S.

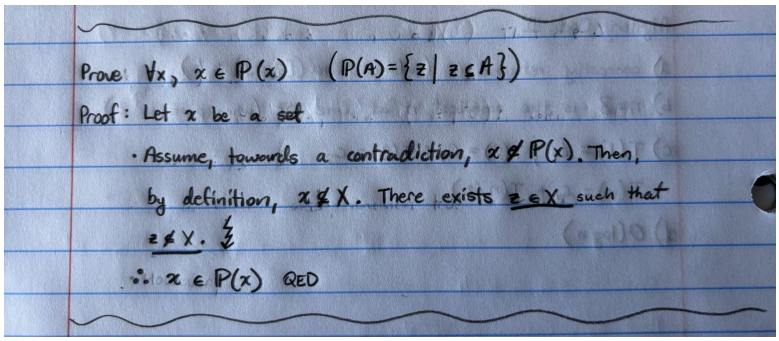
Now we have a problem. A is supposed to be lossless, therefore one-to-one. But A maps a set containing at least $|S| + 1$ files to a set containing $|S|$ files, so the Pigeonhole Principle states that two input files must be mapped to the same output file. This is a contradiction.

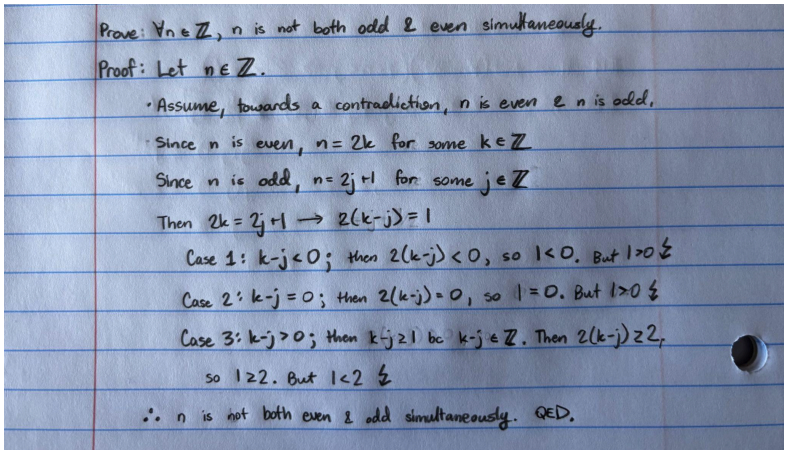
11.11.25 - Lecture

- A contradiction is a false statement
- General outline:
    - We want to prove p.
    - Assume towards a contradiction ¬p
    - We want to find a contradiction (where p and ¬p, which is false)
    - Therefore, p.

Examples:







Prove for any graph with n nodes ($n \geq 2$) that there exists two distinct vertices with the same degree

Proof: Let G be a graph on n nodes where $n \in \mathbb{N}$ & $n \geq 2$.

Let V be the set of vertices of G

Let D be the set of all the degrees of vertices in G

- Consider the function $deg: V \to D$ given by deg(x) = # of edges incident on x
- Observe $|V| = n$. Observe $D \subseteq \{0, 1, 2, \ldots, n-1\}$ so $|D| \leq n$.

Case 1: Suppose there are no nodes w/ degree 0.

Then $D \subseteq \{1, 2, \ldots, n-1\}$, so $|D| \leq n-1$

Case 2: Suppose there is a node x w/ degree 0.

Towards a contradiction, assume there is a node y such that $deg(y) = n-1$. Then there is an edge between y and any other node. So, $deg(x) \geq 1$. However, $deg(x) = 0$ ↯

∴ there is no node of degree n-1. Therefore $D \subseteq \{1, 2, \ldots, n-2\}$, so $|D| \leq n-1$

- So, by the pigeonhole principle, the deg function is not one-to-one, so there exists $v, w \in V$ s.t. $v \neq w$ and $deg(v) = deg(w)$. QED.

Prove for any set x, there is no onto function from x to $\mathcal{P}(x)$

Proof: Let x be a set.

∴ Assume, towards a contradiction, that there exists a function $f: x \to \mathcal{P}(x)$ such that f is onto.

∴ Consider the set $\Delta: \{z \in x \mid z \notin f(z)\}$

Observe $\Delta \subseteq x$, so $\Delta \in \mathcal{P}(x)$

∴ there exists $\delta \in x$ st. $f(\delta) = \Delta$

Case 1: if $\delta \in \Delta$, then $\delta \notin f(\delta)$. Then $\delta \notin \Delta$ bc $f(\delta) = \Delta$. ↯

Case 2: if $\delta \notin \Delta$, then $\neg(\delta \notin f(\delta))$. Then $\delta \in f(\delta)$, so $\delta \in \Delta$ ↯

∴ There is no onto function from $x \to \mathcal{P}(x)$. QED.

Week 13: Countability

20.2 Completeness

- The reals have a property called completeness, which states that any subset of the reals with an upper bound has a smallest upper bound
    - If a sequence of reals converges, the limit it converges to is also real
    - This is not the case for rationals

20.3 Cardinality

- Two sets A and B have the same cardinality iff there is a bijection from A to B
- An infinite set A is countably infinite if there is a bijection from $\mathbb{N}$ (or $\mathbb{Z}$) onto A
    - All subsets of integers are countable

20.4 Cantor-Schroeder-Bernstein Theorem

- $|A| \leq |B|$ iff there is a one-to-one function from A to B
    - This allows us to do very slick 2-way bounding proofs to show a set is countably infinite

20.6 P($\mathbb{N}$) is not countable

- We can represent a subset X of A as a bit vector
    - Example: if $A = \{7, 8, 9, 10, 11\}$ then the bit vector 01100 represents the subset $\{8, 9\}$

	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	...
v_0	1	1	0	1	1	0	1	1	1	1	...
v_1	1	1	0	0	1	0	1	1	0	0	...
v_2	0	0	0	0	1	0	0	1	0	0	...
v_3	0	1	1	1	1	0	1	0	0	0	...
v_4	0	0	0	0	1	1	1	0	1	1	...
v_5	1	1	1	0	1	0	1	0	0	1	...
...	...										

This is supposed to be a complete list of all the bit vectors. But we can construct a bit vector x that's not on the list. The value of x_k, i.e. the kth bit in our new vector, will be 0 if the k digit of v_k is 1, and 1 if the k digit of v_k is 0. Notice that x is different from v_3 because the two vectors differ in the third position. It can't be v_{20} because the two vectors differ in the twentieth position. And, in general, x can't equal v_k because the two vectors differ in the kth position. For the example above, the new vector not in the list would start out: 0 0 1 0 0 1 ....

So, it's not possible to put these infinite bit vectors into a list indexed by the natural numbers, because we can always construct a new bit vector that's not on the list. That is, there can't be a one-to-one function from the infinite bit vectors to the natural numbers. So there can't be a one-to-one function from the subsets of the natural numbers to the natural numbers. So $\mathbb{P}(\mathbb{N})$ isn't countable. That is, the subsets of the natural numbers are more numerous than the natural numbers themselves.

-

20.7 More uncountability results

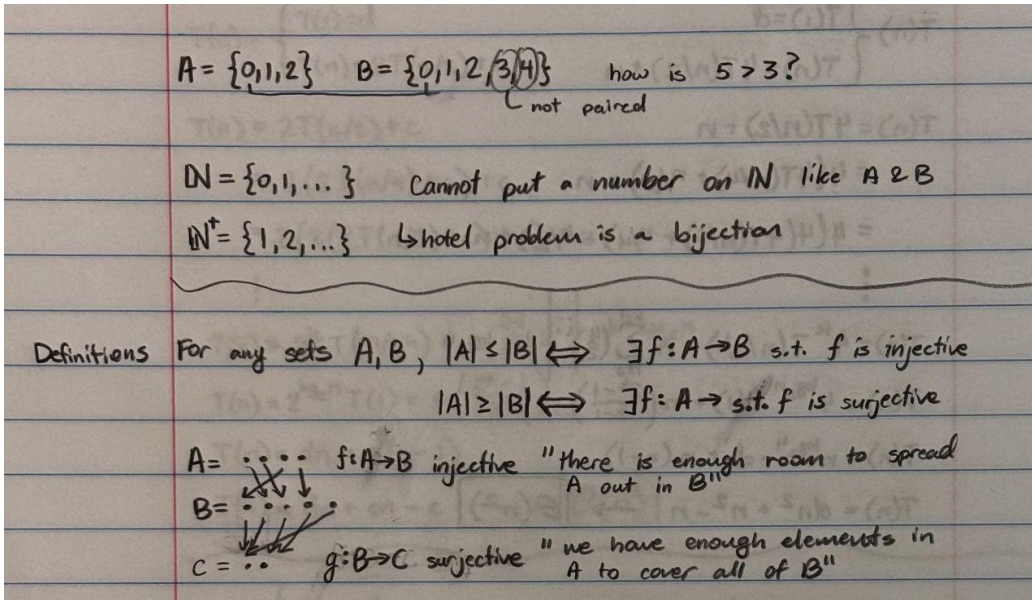
- The reals are not countable

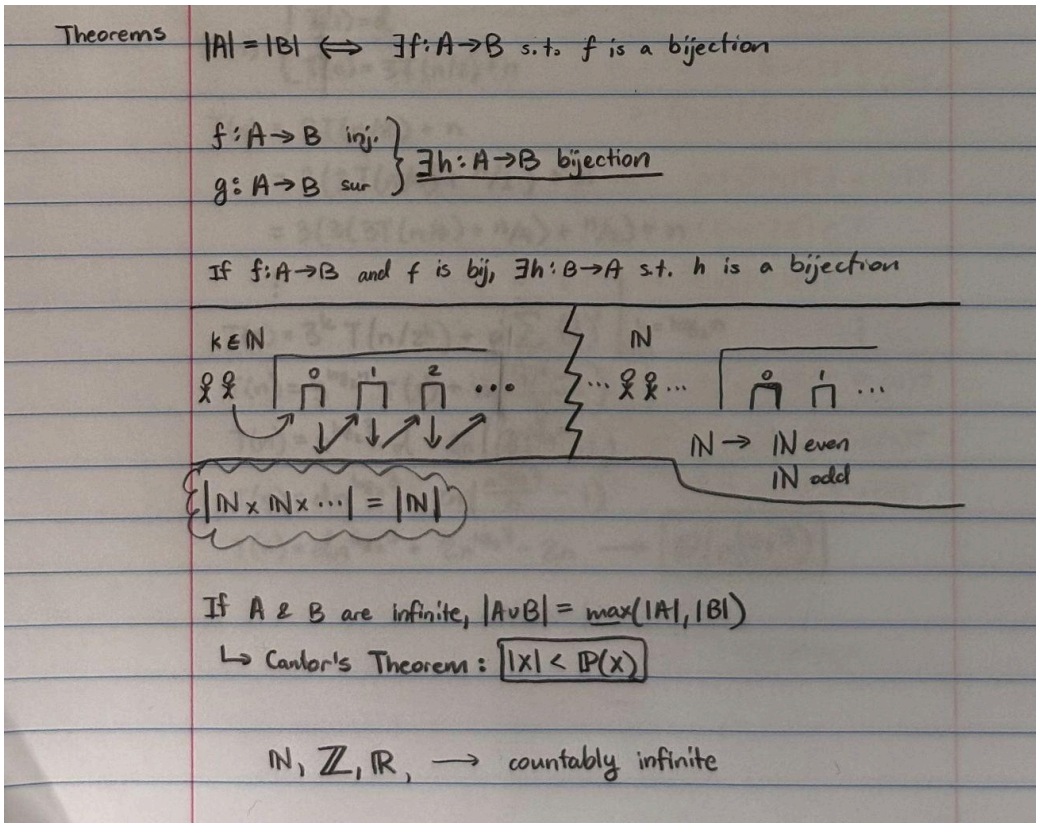
20.8 Uncomputability

- The set of formulas is countable because it is a finite string of characters
    - The set of functions is uncountable (there are many functions) so there are more functions than formulas
- It is not possible to build a program that reads the code of other programs and decides whether they eventually halt or run forever.
    - This is called the Halting Problem
- There are three program behaviors:
    - The program eventually halts so the trace is finite
    - The program loops, so it returns back to a previous state in a sense
    - The program runs forever, consuming more and more storage rather than returning to a previous state

11.18.25 - Lecture

- Some intuition from finite sets will not work on infinite sets
- Fundamental Theorem of Arithmetic - every (natural) number n≥2 has a unique prime factorization



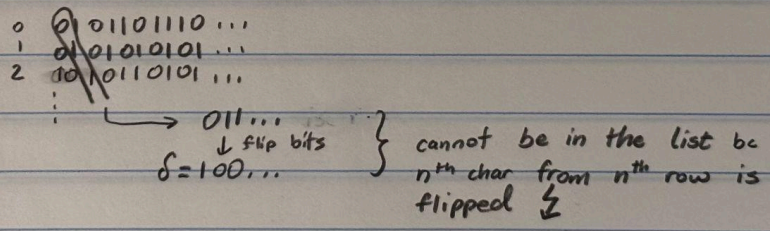


Countable sets $\leq |\mathbb{N}|$ $\quad \mathbb{N} \subsetneq \mathbb{Z}$ (proper subset) but $|\mathbb{N}| = |\mathbb{Z}|$

- A countable union of countable sets is countable

Cantor's Diagonal Argument

Assume $|B_\infty| \leq |\mathbb{N}|$



$\therefore |B_\infty| > |\mathbb{N}|$

Week 14: State Diagrams

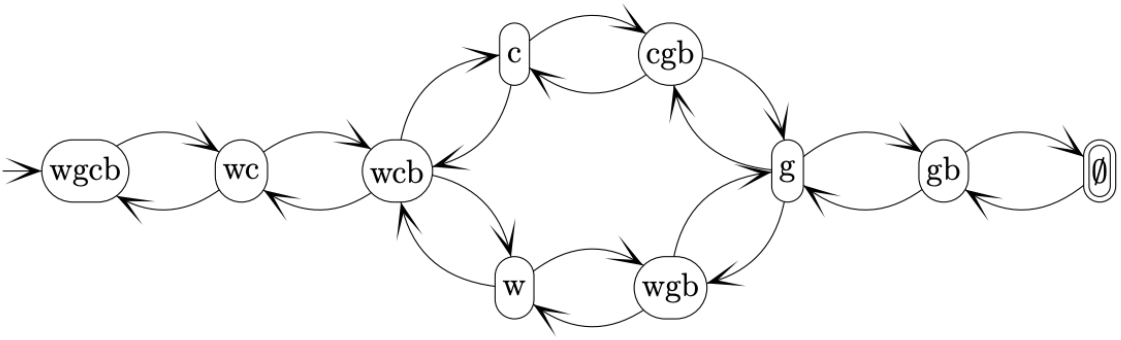
19.1 Introduction

- State diagrams are directed graphs where nodes represent states and labels on the graph represent actions
    - The label on the edge from state A to state B indicates the action that occurs when the system moves from A to B
- Walks must follow arrow directions in state diagrams
    - An action can result in no change in state (self-edge on a node)
    - Two different actions can lead to the same state

19.2 Wolf-goat-cabbage puzzle

State diagrams are often used to model puzzles or games. For example, one famous puzzle involves a farmer taking a wolf, a goat, and a cabbage to market. To do this, he must cross from the east to the west side of a river using a boat that can only carry him plus one of his three possessions. He cannot leave the wolf and goat together unsupervised, nor the goat and the cabbage, because one will eat the other.

We can represent each state of this system by listing the objects that are on the east bank: w is the wolf, g is the goat, c is the cabbage, and b is the boat. States like wc and wgb are legal, but wg would not be a legal state. The diagram of legal states then looks as follows:

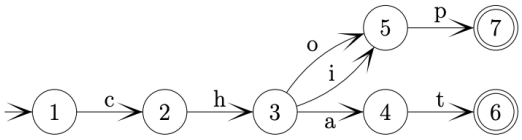


In this diagram, actions aren't marked on the edges: you're left to infer the action from the change in state. The start state (wgcb) where the system begins is marked by showing an arrow leading into it. The end state (∅) where nothing is left on the east bank is marked with a double ring.

- There are two shortest solutions, but an infinite number of other solutions (there is a loop in the problem)

19.3 Phone lattices

- In phone lattices, each edge represents the action of reading a single sound from the input stream of speech
    - Ex: we can model the set of words {chat, chop, chip} as



- Occasionally, it is useful to have a self-loop for multiple occurrences of the same letter

19.4 Representing functions

- A function from {a,b,c} to {1,2,3,4} can be represented as {(a,4), (b,1), (c,4)}

19.5 Transition functions

- We can make transition functions to map out the edges of a graph
    - This is represented as $\delta = S \times A \rightarrow P(A)$

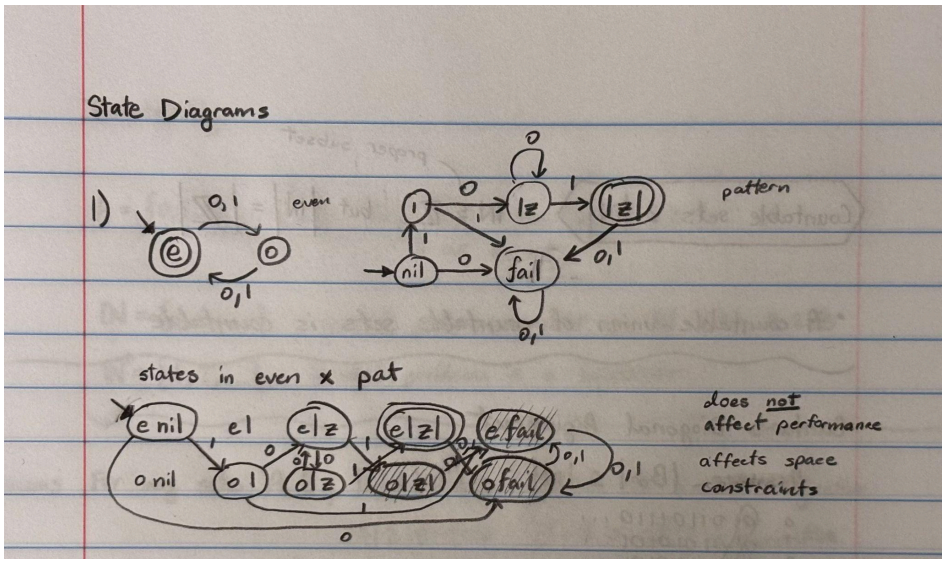
19.6 Shared states

- Overlapping words can be condensed into a more compact phone lattice
- Dynamic programming allows us to use the results of previous work rather than repeating them, which significantly improves the running time

12.2.25 - Lecture

Problem 1. Simple patterns

Suppose that our set of input characters contains 0 and 1. Let's build DFAs to recognize two simple patterns: even length strings and strings that have exactly the form $1(00)^+1$.

- 

Problem 3. WYSWYG

We want to build a DFA that will recognize if the string '**WYSWYG**' appears anywhere in a sequence of characters. For simplicity, let's assume that no other characters will appear except these four.

-