CSci 243 Discrete Structures, Lecture 1

Tools for mathematical reasoning :

Logic ($\lambda o \gamma o s$: art of reasoning) + discrete structures (countable)

Logic and Proofs

Truth : Contrast to Truth in philosophy which has different meanings:

- corresponding to facts (empirical sciences)
- formed by society
- existing outside humans (platonic ideas)
- Coherence

  Set of Principles considered true (axioms) $\Longrightarrow$ deduction tools $\Longrightarrow$ Theorems

Here we focus on a coherent system = set of all truths (axioms + theorems)
Logic is essential in **any** formal discipline and consists of rules for drawing inferences. Since logic can be **separated from the context of the discipline** to which it is applied, it provides a suitable tool for reasoning.
When applied to computer science, logic is used to give precise meaning to mathematical statements and to establish rigorous proofs, such as the proofs of correctness of algorithms.

Propositional and predicate logic *Reading: Rosen 1.1 – 1.5*

Natural languages are plagued by ambiguity (a necessary fact in humans)
Definition: A **proposition** is a statement that has a **truth value**; it is either **true** (T) or **false** (F), but not both. Lowercase letters, such as p, q, r, s, are often used to represent propositions.

Examples

”How much does it cost?” (not)
”This painting is ugly” (not, this is subjective)
”the radius of the earth is 1Km” (yes)
”1+1=2”, ”1+1=3”. (yes)
”x+1 = 2” (not quite. Depends on value of x)

Definition: Logic dealing with propositions is called Propositional Calculus

Compound propositions can be constructed by using one or more **logical operators**.

- Unary operator: $\neg$ (not) (altern. $\bar{p}$)
- Common binary operators: $\wedge$ (and, conjunction), $\vee$ (or, disjunction)
- Other binary operators: $\oplus$ (exclusive-or),
  $\rightarrow$ (implication, conditional), $\leftrightarrow$ (equivalence, biconditional)

The following truth tables **define** these logic operators.

p	$\neg p$
F	T
T	F

p	q	$p \wedge q$	$p \vee q$
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

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

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

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

Remarks:

- In a truth table, F and T can also be written as 0 and 1, respectively.
- Precedence order of operators: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. Parentheses can be used to override the order.
- Two compound propositions are equivalent ($\equiv$) or equal ($=$) if they have the same truth table. For example,

$$p \oplus q \equiv \quad (p \wedge \neg q) \vee (\neg p \wedge q) \quad \equiv \quad (p \vee q) \wedge (\neg p \vee \neg q)$$

"Either the car is a Ferrari OR the car color is red" $\equiv$
"(car is a Ferrari AND color Not red) OR (car is Not a Ferrari AND color is red)"
$\equiv$
"(car is a Ferrari OR color is red) AND (car is Not a Ferrari OR color is Not red)"

Complicated in every day language: Better to use Propositional calculus

- $$p \rightarrow q \equiv \quad \neg q \rightarrow \neg p \quad \equiv \quad \neg p \vee q$$

  p = car is a Ferrari, q = car is red. $p \rightarrow q$ means:
  Ex A: $\neg q \rightarrow \neg p \equiv$ "If not red, car is not a Ferrari"
  Ex B: $\neg p \vee q \equiv$ "Not a Ferrari OR a car is red"
  $p \rightarrow q$ can also be read as

  - if p then q, p implies q, q follows from p, q whenever p
  - p is sufficient for q, a sufficient condition for q is p
  - q is necessary for p, a necessary condition for p is q
  - p only if q, q unless $\neg p$

  p is called the hypothesis, or the premise, or the antecedant
  q is called the conclusion or consequence

- The equivalence of $p \rightarrow q$ and $\neg q \rightarrow \neg p$ establishes the correctness of the proof by contradiction method.

- It may be hard to digest that $F \rightarrow T$ is a true implication! Think that this is not about the outcome (the conclusion) or even about the premise. It is about the process of "implication". If the hypothesis doesn't hold, we do not care what lies in the conclusion. E.g., we do not care what the q is inside an "if p then q" statement because it will not execute if p is false.

- Mentally think of $p \rightarrow q$ as p being whether a spigot is on or off and q whether water is running in the yard. If $p = T$ then $q = T$. But if $p = F$, water can still be running from another source. Similarly with the Ferrari implying a red car example, does not mean that if I see a red car this is a Ferrari.

- The definition of $\rightarrow$ is through its truth table, not through the English *if, then*. "The above intuitive explanations are supposed to convince you (or not) that it is reasonable to use those two English words to speak about logical implication, not that logical implication ought to work that way in the first place." (Henning Makholm)

- Note also that $F \to F$ must be true so that we can define the equivalence appropriately afterwards. On the other hand, if we set $F \to T = F$ then implication and equivalence would be identical.
- When used with quantifiers (see later), the implication

  $$\forall x \in Q(x \text{ is rational } \to x = \text{ ratio of integers})$$

  should be true even if Q is only the natural numbers!
- $$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$$

  This is read as
  - p and q are equivalent
  - p is necessary and sufficient for q
  - p if and only if q
  - p iff q

  Eg., "Car is a ferrari iff it is red"
- $\neg, \wedge, \vee$, these three operators are sufficient to define **all** compound propositions.

Lecture 2

Propositional functions:
For example, $\phi(p,q) = (p \vee q) \wedge (\neg p \vee q)$. Similar to calculus, terms such as variable and expression/formula or function, are also used here.

For each propositional function, there is a truth table. Vice versa, any truth table defines a propositional function.

For a n-dimensional function with n different propositional variables, there are 2^n entries (rows) in the truth table. This is because we need to have all possible bit strings of (*****) of length n.

Since a propositional function with n variables has to define a column of exactly 2^n rows with 0/1, we have exactly 2^{2^n} different possible functions.

Example, There are $2^{2^1} = 4$ unary operators, $2^{2^2} = 16$ binary operators, etc,...

The number of rows grows exponentially, and the functions superexponentially, so proving equivalences through truth tables becomes very time consuming. Better to use algebraic laws:

- Identity law: $p \wedge \mathrm{T} \equiv p$, $p \vee \mathrm{F} \equiv p$
- Domination law: $p \vee \mathrm{T} \equiv \mathrm{T}$, $p \wedge \mathrm{F} \equiv \mathrm{F}$
- Idempotent law: $p \vee p \equiv p$, $p \wedge p \equiv p$
- Double negative law: $\neg(\neg p)) \equiv p$
- Negation law: $p \vee \neg p \equiv \mathrm{T}$, $p \wedge \neg p \equiv \mathrm{F}$
- Absorption law: $p \vee (p \wedge p) \equiv p$, $p \wedge (p \vee p) \equiv p$
- Commutative law: $p \vee q \equiv q \vee p$, $p \wedge q \equiv q \wedge p$
- Associative law: $(p \vee q) \vee r \equiv p \vee (q \vee r)$, $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
- Distributive law: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$, $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
- De Morgan's law: $\neg(p \vee q) \equiv \neg p \wedge \neg q$, $\neg(p \wedge q) \equiv \neg p \vee \neg q$
  ex: "Not true that I am old and rich" = "I am not old OR I am not rich"
  ex: $\neg(p \to q) \equiv \neg(\neg p \vee q) \equiv p \wedge \neg q$
  ex: ¬(I am student $\to$ I go to school) $\equiv$ I am student AND donot go to school

Definition. A **predicate** is a statement containing variables; it is either true or false depending on the values assigned to the variables. Upper case letters are usually used to represent predicates, e.g., $P(x) =$ "$x > 3$" and $Q(x,y) =$ "$x+y=0$".

- Logical operations defined for propositions can all be applied to predicates to get compound predicates.
- A predicate can involve more many variables: n-place predicate or n-ary predicate $P(x_1,\ldots,x_n)$.
  For a particular tuple, the value of the propositional function is the n-place predicate.
  Example:
  if $(x \neq 0)$, then $\frac{1}{x} = \frac{1}{x^2}$, if $P(x) \to Q(x)$
  $(x<0) \wedge (y>0) \to (xy<0)$
- There are further logical operators, known as **quantifiers**, which when applied to a predicate to produce a true proposition or a false proposition.
  - Universal quantifier: $\forall$ (for all), $\forall x P(x)$
    Example: $\forall x(x>3)$, which is false.
    Example: "All humans are mortals" (ie., $\forall x$ human, x is mortal).
  - Existential quantifier: $\exists$ (there exists), $\exists x P(x)$
    Example: $\exists x(x>3)$, which is true.
    Example: "There is a human with six fingers"
- De Morgan's law with quantifiers

$$\neg \forall x P(x) \equiv \exists x(\neg P(x))$$

$$\neg \exists x P(x) \equiv \forall x(\neg P(x))$$

  Examples: $\neg \forall x(x>3) \equiv \exists x(x \leq 3), \qquad \neg \exists x(x>3) \equiv \forall x(x \leq 3)$
- Quantified predicates are useful in stating mathematical results, such as $\forall x \exists y P(x,y)$ (for all x there is y such that) and $\exists y \forall x P(x,y)$ (there is y such that for all x).

  Example 1: $\forall x \exists y(x+y=0)$, which is a true statement.

  Example 2: $\exists y \forall x(x+y=0)$, which is a false statement.
- Instead, in $\forall x \forall y$ and $\exists x \exists y$ the order does not matter
- In a sequence of quantifiers, apply De Morgan's law from outside to the inside:

$$\neg \forall x \forall y P(x,y) \equiv \exists x \neg(\forall P(x,y)) \equiv \exists x \exists y \neg P(x,y)$$

Lecture 3

Rules of inference

Reading: Rosen 1.5 – 1.6

Argument = a sequence of statements that ends with a conclusion

Valid argument means: if premises (preceding statements) are true, then the conclusion (final statement) must be true.

Typically argument = sequence of propositions
argument form = sequence of compound propositions or predicates

We write:

proposition
...
proposition

Conclusion

Rules of Inference

1. Modus Ponense $\dfrac{p \quad p \to q}{q}$ Example: $\dfrac{\text{I have a Ferrari} \quad \text{car is a Ferrari implies that car is red}}{\text{I have a red car}}$

2. Modus Tollens $\dfrac{\neg q \quad p \to q}{\neg p}$ Example: $\dfrac{\text{I don't have a red car} \quad \text{car is a Ferrari implies that car is red}}{\text{I don't have a Ferrari}}$

3. Hypothetical Syllogism $\dfrac{p \to q \quad q \to r}{p \to r}$ Example: $\dfrac{\text{all Ferraris are red} \quad \text{red cars get tickets}}{\text{Ferraris get tickets}}$

4. Disjunctive Syllogism $\dfrac{p \vee q \quad \neg p}{q}$ Example: $\dfrac{\text{I am old or rich} \quad \text{I am not old}}{\text{I am rich}}$

5. Addition $\dfrac{p}{p \vee q}$ If p for sure $p\vee$ anything!

6. Simplification $\dfrac{p \wedge q}{p}$ Example: $\dfrac{\text{I am old and rich}}{\text{I am old}}$

7. Conjunction $\dfrac{p \quad q}{p \wedge q}$ States that all premises are ANDed

8. Resolution $\dfrac{p \vee q \quad \neg p \vee r}{q \vee r}$ Powerful deductive tool in functional/logic languages (e.g., Prolog)

   Example: $\dfrac{\text{My car is a Ferrari or an Audi} \quad \text{My car is not a Ferrari or it is red}}{\text{My car is Audi or Red}}$ (I could still have a Ferrari, or a red Audi)

   Example: $\dfrac{\text{I read a book or went to the movies} \quad \text{I did not read a book or went for dinner}}{\text{I went to the movies or for dinner}}$

Tautology: a propositional function that is always true. Rules of inference are tautologies

Example: $(p \wedge (p \rightarrow q)) \rightarrow q$ is the tautology of "modus ponense"!

You can prove this inference rules by constructing the truth tables.
Alternatively, use propositional calculus. Example for showing the modus ponense:

$$\begin{aligned}
(p \wedge (p \rightarrow q)) \rightarrow q &\equiv (p \wedge (\neg p \vee q)) \rightarrow q && \text{replacing } \rightarrow \\
&\equiv (p \wedge \neg p) \vee (p \wedge q) \rightarrow q && \text{distributive} \\
&\equiv (F \vee (p \wedge q) \rightarrow q && \text{Identity} \\
&\equiv \neg(p \wedge q) \vee q && \text{replacing } \rightarrow \\
&\equiv \neg p \vee \neg q \vee q && \text{De Morgan} \\
&\equiv \neg p \vee T \equiv T && \text{negation}
\end{aligned}$$

FALLACIES: Using "rules" that resemble tautologies as inference rules. However these are based on contingencies so they are not always true!

(Remember correlation does not mean causation!)

Fallacy of affirming the conclusion: $((p \to q) \wedge q) \to p$
Note that if $p = F, q = T$, the above is F, so not a tautology!
Example: we know it is true that:
"If I download illegal music, then there is music on my hard drive".
Assume now that "there is music on my hard drive".
Does it mean that I downloaded illegal music? NOT!

Fallacy of denying the hypothesis: $((p \to q) \wedge \neg p) \to \neg q$
Common mistake when proving by contradiction or contraposition
Example: "If I download illegal music, then there is music on my hard drive".
Also "I did not download illegal music".
Can I conclude that "There is no music on my hard drive"? NOT!

Rules of inference for quantified statements
Universal instantiation $\forall x P(x) \to P(c)$ for any specific c
Universal generalization $P(c)$ for an arbitrary c (i.e., with no specific properties) $\to \forall x P(x)$
Existential instantiation $\exists x P(x) \to P(c)$ for some specific c
Existential generalization $P(c)$ for some $c \to \exists x P(x)$

Universal Modus Ponense
$$\begin{array}{l} \forall x(P(x) \to Q(x)) \\ P(c) \\ \hline Q(c) \end{array}$$

Universal Modus Tollens
$$\begin{array}{l} \forall x(P(x) \to Q(x)) \\ \neg Q(c) \\ \hline \neg P(c) \end{array}$$

Lecture 4

Methods of proofs *Reading: Rosen 1.7–1.8*

Proof = sequence of statements, where each statement is given condition or conclusion obtained by previous statements by some rules of infererence

Argument:	Proof:
proposition	statement
statement	$\rightarrow$ conclusion 1
...	$\rightarrow$ conclusion 2
proposition	...
_____	$\rightarrow$ final conclusion
Conclusion	

Methods of proof

- **Direct proof:** To prove "if p then q", assume p is true and show that q is true. (we only need to show this, since if $p = F$ the implication always true).

  Example: Prove that if m and n are odd numbers, so is mn.

  Proof: Since m and n are odd numbers, then by the definition of odd numbers, m can be written as $m = 2a+1$ for some integer a, and n written as $n = 2b+1$ for some integer b. Then $mn = (2a+1)(2b+1) = 4ab+2a+2b+1 = 2(2ab+a+b)+1$. Since $mn = 2c+1$, where $c = 2ab+a+b$ is an integer, then by the definition of odd numbers, mn is an odd number.

- **Proof by contraposition:** To prove "if p then q", assume q is false and show p is false. This method is based on $p \rightarrow q \equiv \neg q \rightarrow \neg p$.

  Note this is not quite Modus Tollense $((\neg q \wedge p \rightarrow q) \rightarrow \neg p)$ because we do not want to assume $p \rightarrow q$. We want to prove it.

  Example: Prove that if $n = ab$, where a and b are positive integers, then $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$.

  Proof: Assume the opposite is true, i.e., $a > \sqrt{n}$ and $b > \sqrt{n}$. Then $ab > \sqrt{n} \cdot \sqrt{n} = n$. So $n \neq ab$, which contradicts the given condition $n = ab$.

- **Proof by contradiction:** To prove "p", assume p is false and show there is r such that r is true and $\neg r$ is true, a contradiction. This method is based on $p \equiv \neg p \rightarrow (r \wedge \neg r)$.

  Example: Prove that $\sqrt{2}$ is an irrational number.

Proof: Assume $\sqrt{2}$ is not irrational, thus is rational. Then by the definition of rational numbers, there are two integers a and b with $gcd(a,b) = 1$ such that $\sqrt{2} = \frac{a}{b}$. Squaring both sides of the equation, we get $2 = \frac{a^2}{b^2}$, thus $2b^2 = a^2$. So a must be an even number, or $a = 2c$ for some integer c. Then the equation $2b^2 = a^2$ becomes $2b^2 = (2c)^2 = 4c^2$, thus $b^2 = 2c^2$. Since b^2 is an even number, b must be an even number. So both a and b are even, thus $gcd(a,b) \neq 1$. A contradiction to $gcd(a,b) = 1$.

Note: The difference between contraposition and contradiction is subtle. In doing proofs, people usually don't distinguish the two methods, and call both "proof by contradiction".

Contraposition shows $p \to q$
Contradiction shows p
To use contradiction to show $p \to q$, assume $\neg(p \to q) = p \wedge \neg q$, and show $\neg p$. Then we have $p \wedge \neg p$ which is contradiction.

- **Proof by cases:** When we are unable to prove a theorem with a single argument that hold for all possible cases, we may need to construct our proof for each of the possible cases, one by one. The rule of inference behind this method lies on $(p_1 \vee p_2 \vee \cdots \vee p_n) \to q \equiv (p_1 \to q) \wedge (p_2 \to q) \wedge \cdots \wedge (p_n \to q)$.

  Example: Show that $|x \cdot y| = |x| \cdot |y|$.

  Proof: We consider the following cases:

  (1) $x \geq 0$ and $y \geq 0$: $|x \cdot y| = x \cdot y = |x| \cdot |y|$.

  (2) $x \geq 0$ and $y < 0$: $|x \cdot y| = -x \cdot y = x \cdot (-y) = |x| \cdot |y|$.

  (3) $x < 0$ and $y \geq 0$: $x \cdot y| = -x \cdot y = (-x) \cdot y = |x| \cdot |y|$.

  (4) $x < 0$ and $y < 0$: $|x \cdot y| = x \cdot y = (-x) \cdot (-y) = |x| \cdot |y|$.

  Example: Show that there are no integer solutions to equation $x^2 + 3y^2 = 8$.

  Proof: Observing the equation, we get $x^2 \leq 8$ and $3y^2 \leq 8$. Since x and y are integers, then $x = 0, \pm 1, \pm 2$ and $y = 0, \pm 1$. This leaves us with only 5×3 possible combinations of x and y to check.

- **Existence proof:** To prove there exists an object of a certain type, we can simply construct one. This is the **constructive existence proof**.

  Example: Prove that there is a positive integer $(\exists n \in \mathbb{Z}_+)$ that can be written as the sum of cubes of positive integers in two different ways.

  Proof: $1729 = 10^3 + 9^3 = 12^3 + 1^3$.

It is also possible to give a **nonconstructive existence proof**. Sometimes a nonconstructive existence proof uses proof by contradiction.

Example: Prove that there exist irrational numbers x and y such that x^y is rational.

Proof: Consider the irrational number $\sqrt{2}$ and make a number $z = \sqrt{2}^{\sqrt{2}}$. If z is a rational number, then we have found x and y. However, if z is irrational, then let $x = z$ and $y = \sqrt{2}$ so that $x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$, a rational!

Note that I do not know which is the case, but one of the two must be true.

- **Proof by induction:** To prove that $P(n)$ is true for all $n \geq 1$, prove that

  1. $P(1)$ is true and
  2. $\forall k(P(k) \rightarrow P(k+1))$ is true.

  Example: Prove that $\sum_{i=1}^{n} = 1+2+\cdots+n = \frac{1}{2}n(n+1)$ for all $n \geq 1$.

  Proof: Let $P(n)$ be $1+2+\cdots+n = \frac{1}{2}n(n+1)$. We induct on n.

  Basis step: $P(1)$ is obviously true, because $1 = \frac{1}{2} \cdot 1 \cdot (1+1)$.

  Inductive step: Assume $P(k)$ is true, i.e., $\sum_{i=1}^{k} = 1+2+\cdots+k = \frac{1}{2}k(k+1)$. We will show that $P(k+1)$ is also true, i.e., $\sum_{i=1}^{k+1} = 1+2+\cdots+(k+1) = \frac{1}{2}(k+1)(k+2)$.

$$\begin{aligned} 1+2+\cdots+(k+1) &= (1+2+\cdots+k)+(k+1) \\ &= \frac{1}{2}k(k+1)+(k+1) \\ &= \frac{1}{2}(k+1)(k+2) \end{aligned}$$

  Note: We cover also some material from Chpt 5, but will revisit with recursion

  Example: If at step 0 we have the number 1, and then we double it every step, prove that at step k the number is 2^k.

  Basis: $k = 0$, $2^0 = 1$ true.
  IH: Assume for k, number is 2^k
  IS: For $k+1$, it is double the previous one, so $2 * 2^k = 2^{k+1}$.

Lecture 5 (see also Chapter 5)

Example: Prove that for any positive integer n, n^3+2n is divisible by 3.

Proof. $P(n) = n^3+2n$ is divisible by 3

Basis: $n = 1$, $P(1) =$ 1+2 is divisible by 3. True.

IH: Assume (k^3+2k is divisible by 3).

IS: Show that $((k+1)^3+2(k+1)$ is divisible by 3).

$$(k+1)^3+2(k+1) = k^3+3k^2+5k+3$$
$$= (k^3+2k)+(3k^2+3k+3) \text{ then from IH:}$$
$$= 3M+3(k^2+k+1) = \text{ multiple of 3 .}$$

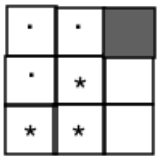
Induction Example: Trominoes: Tromino is an L-shaped figure of three squares:

* Consider $n \times n$ grid and a pile of trominoes.

* Pick any square of the grid and cut a hole in it (not to be used)

* Question: Can we tile the entire grid (except the selected tile) with trominoes?

Answer in general: No!



Take e.g., the 3x3 grid. There are 8 tiles to be filled with 3-tile trominoes. However, 8 is not divisible by 3! (see figure)

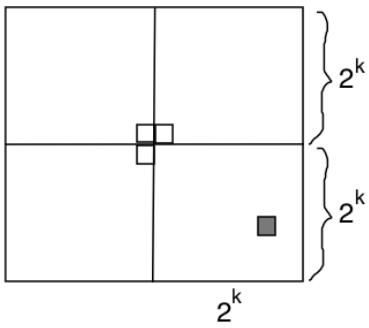
Prove that when $n = 2^k$, $k = 1,2,\ldots$, the tiling is possible.

Proof. By induction on k (not on n):

Base case: $k = 1$, the grid is $n = 2$ and I can trivially use 1 tile:

I.H.: Assume that it is true for k, i.e., for any $n \times n$ grid of $n = 2^k$

I.S.: Consider a grid of size $n = 2^{k+1}$. Since $2^{k+1} = 2 * 2^k$, I can split the grid into four subgrids, each of size $2^k \times 2^k$:



* Consider the hole to be in any of the four subgrids (WLOG say in the (2,2) subgrid as shown in the picture). This subgrid is of dimension 2^k and thus by I.H. it is tileable with trominoes.

* For the rest three subgrids, insert 3 black squares where the subgrids meet. One square per subgrid as shown in the center of the figure. Note that one trominoe fits in these three spots.

* By I.H. all three subgrids (size 2^k) are tileable

*Consider all the trominoes in the four subgrid tilings, and add the center one. Then this consistitues a tiling of the 2^{k+1} grid.

Induction Example: Game of hats
* In a room there m people, all of them wearing a hat
* People wear either a blue or a red hat and cannot see their own hat, only other people's.
* The number of blue hats is $n \geq 1$.
* Every few seconds a bell rings at which time everyone who knows (i.e., can logically deduce) the color of their hat must call it out loud. No other exchange is allowed.
Prove that all those with blue hats will call it at exactly the n-th call.
Proof
Base case: $n = 1$, i.e., one person wears a blue hat.

Person wears:	b	r r $\cdots$ r
Person thinks how many blue hats there can be:	1	1 or 2

Note that the person wearing blue knows it is her since n cannot be zero. So at the first ring, she will call it.

I.H.: Assume that for $n = k$ it holds, i.e., all those with k blue hats will call it at the k-th ring.
I.S.: Assume there are $n = k+1$ blue hats.
Every person with a blue hat sees k blue hats in the crowd, so (s)he thinks there can be k or $k+1$ blue hats in total.
Every person with a red hat sees $k+1$ blue hats in the crowd, so (s)he thinks there can be $k+1$ or $k+2$ blue hats in total.
Now the blue hat people think that if there were k hats in total, by I.H., the crowd would have called it at the k-th step. If they didn't hear anyone calling at the k-th step, then there must be $k+1$ hats, so they will call it at the $k+1$ step. QED

* Note: If everyone knows in addition that there is at *least one red hat*, then the algorithm should stop in $\min(m-n, n)$ rings. This is because if the number of red hats is smaller than the blue ones, then those wearing red hats will call it out first.

** Note: This strategy breaks down if the people do not know that one of the colors must be ≥ 1. Then, a one-person crowd can never know the color. Similarly, two people can have (bb)(br)(rb) or (rr). In all cases, everyone sees at least two options, and additional rings do not change this.

Basic Structures: Sets/Functions/Sequences/Sums, Lecture 6

Sets *Reading: Rosen 2.1 and 2.2*

Set theory is one of the cornerstones of mathematics and provides a convenient language for describing concepts in mathematics and computer science (also logic, philosophy..)

A **set** is an **unordered** collection of **elements**. $x \in Y$ means element x belongs in set Y.

In 1895 Cantor gave the 1st definition of sets and developed theory. However, that theory leads to paradoxes and newer axiomatic definitions have been more successful. Still, they are similar to Cantor's for most of the common uses of sets.

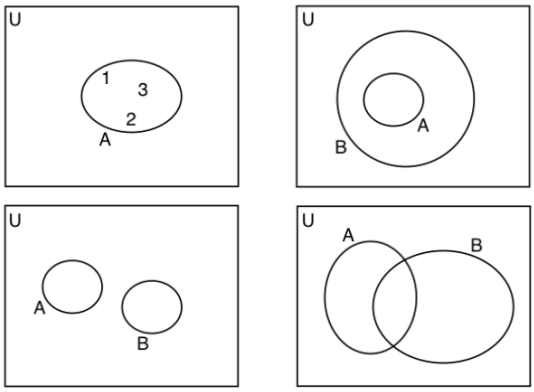
Example of Barber's paradox: A barber is the person that shaves all men who do not shave themselves and only men who do not shave themselves. So who shaves the barber? As the barber, he should not shave himself. But as a barber, he should shave all men that do not shave themselves, i.e., shave himself.

Russel's paradox: Let $Y = \{x | x \notin x\}$. Then, $Y \in Y \leftrightarrow Y \notin Y$.

Basic definitions for sets:

- Uppercase letters are used to name sets and lowercase letters to name set elements. Examples of sets are $S = \{1,2,3\}$, $A = \{x | x \text{ is an even integer}\}$.
- Some special sets of numbers: $\mathbb{N}$ (set of natural numbers), $\mathbb{Z}$ (set of integers), $\mathbb{Q}$ (set of rational numbers), $\mathbb{R}$ (set of real numbers). Superscript or subscript $^+$ is used for positive numbers. Superscript * means the set does not include the zero.
- The **empty set** is a set without any element, denoted by $\emptyset$ or $\{\}$. Note that $\emptyset \neq \{\emptyset\}$. A **singleton** is a set of only one element (e.g., the $\{\emptyset\}$).
- $A \subseteq B \leftrightarrow \forall x \in A(x \in B)$. A is a **subset** of B if every element of A is also an element of B.
- $A = B \leftrightarrow A \subseteq B \wedge B \subseteq A$. Sets A and B are said to be **equal**.
- $A \subset B \leftrightarrow A \subseteq B \wedge A \neq B$, A is called a proper subset of B
- A **Venn diagram** can be used to represent the subset relation of two sets.

Set A, $A \subset B$, and two different ways for $A \neq B$



- A set may be finite or infinite, depending on whether the number of elements it contains is finite or infinite. $|S|$ is the **cardinality** of a finite set S and is defined to be the number of elements in the set.
  Ex: $|\emptyset| = 0, |\{a,b\}| = 2, |\{\emptyset\}| = 1$. For infinite sets, extension possible but with care!

- For a given set S, the **power set** of S, $P(S)$ or 2^S, is the set of all subsets of S. For example, if $S = \{0,1,2\}$, then $P(S) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\}, \{1,2\}, \{0,1,2\}\}$.

  $|2^S| = 2^{|S|}$

  e.g., with every new element we double the sets that can be produced!

- For two sets A and B, the **Cartesian product** of A and B, denoted by $A \times B$, is the set of all ordered pair from A and B, i.e., $A \times B = \{(a,b) | a \in A \wedge b \in B\}$.
  Ex: $A = \{z,x\}$, $B = \{$ \$25, \$50, \$100 $\}$, then
  $A \times B = \{(z,\$25), (z,\$50), (z,\$100), (x,\$25), (x,\$50), (x,\$100\}$

  Note that $B \times A \neq A \times B$.

  $|A \times B| = |A||B|$

- Cartesian product of many sets:
  $$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, \ldots, a_n) | a_i \in A_i, i = 1, \ldots n\}$$
  The ordered $(a_1, \ldots, a_n)$ is called n-tuple
  Ex: $\{a\} \times \{1,2\} \times \{Z,Q\} = \{(a,1), (a,2)\} \times \{Z,Q\} =$
  $\{(a,1,Z), (a,2,Z), (a,1,Q), (a,2,Q)\}$

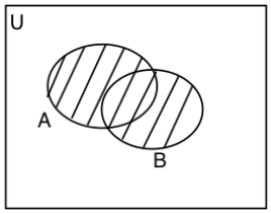
- A subset of $A \times B$ is called **relation** from A to B (functions are special relations)

- It is useful to restrict quantifiers to particular sets,
  e.g., $\forall x \in S(P(x))$, $\exists x \in R(Q(x))$, $\forall x \in S \exists y \in F(P(x,y))$

- Truth Set of a predicate $P(x)$ is defined as $\{x \in D | P(x)\}$

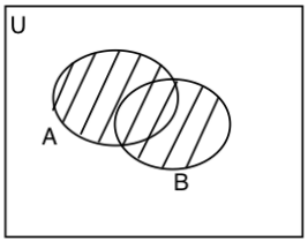
Set operations:

- **Union**: $A \cup B = \{x | x \in A \vee x \in B\}$, $\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n$



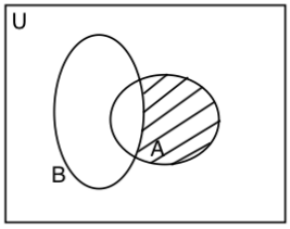
- **Intersection**: $A \cap B = \{x | x \in A \wedge x \in B\}$, $\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \cdots \cap A_n$

  A and B disjoint iff $A \cap B = \emptyset$.
  $|A \cup B| = |A| + |B| - |A \cap B|$
  (Why? Note that adding the cardinalities counts the elements of $A \cap B$ twice!)

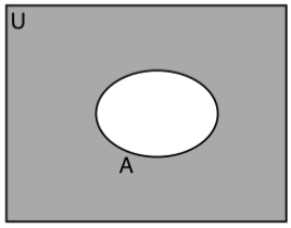


- **Difference**: $A - B = \{x | x \in A \wedge x \notin B\}$

  Clearly: $|A - B| = |A| - |A \cap B|$



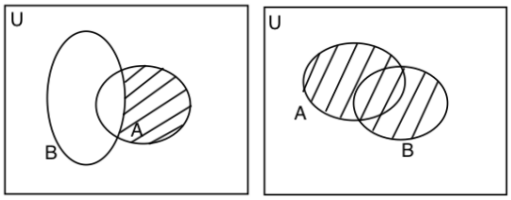
- **Complement** with respect to a **universal set** U: $\overline{A} = U - A = \{x | x \notin A\}$



Set identities are used to simplify or manipulate an expression that involves sets.

- Identity law: $A \cup \emptyset = A$, $A \cap U = A$
- Domination law: $A \cup U = U$, $A \cap \emptyset = \emptyset$
- Idempotent law: $A \cup A = A$, $A \cap A = A$.

- Complementation law: $\overline{(\overline{A})} = A$
- Absorption law: $A \cup (A \cap B) = A$, $A \cap (A \cup B) = A$



- Complement law: $A \cup \overline{A} = U$, $A \cap \overline{A} = \emptyset$
- Commutative law: $A \cup B = B \cup A$, $A \cap B = B \cap A$
- Associative law: $A \cup (B \cup C)) = (A \cup B) \cup C$, $A \cap (B \cap C) = (A \cap B) \cap C$
- Distributive law: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$, $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- De Morgan's law: $\overline{A \cup B} = \overline{A} \cap \overline{B}$, $\overline{A \cap B} = \overline{A} \cup \overline{B}$

There are two ways to prove the equality of two set expressions. One is to use the definition of equal sets plus the above set identities. The other is to use the so-called **membership table**.

Example: Prove the first De Morgan's law that $\overline{A \cup B} = \overline{A} \cap \overline{B}$.

Proof 1. To prove $\overline{A \cup B} = \overline{A} \cap \overline{B}$, we prove (1) $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$ and (2) $\overline{A} \cap \overline{B} \subseteq \overline{A \cup B}$.

(1) $\forall x \in \overline{A \cup B} \Rightarrow x \notin A \cup B \Rightarrow \neg(x \in A \cup B) \Rightarrow \neg(x \in A \vee x \in B) \Rightarrow \neg(x \in A) \wedge \neg(x \in B) \Rightarrow x \notin A \wedge x \notin B \Rightarrow x \in \overline{A} \wedge x \in \overline{B} \Rightarrow x \in \overline{A} \cap \overline{B}$. Therefore, $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$.

(2) $\forall x \in \overline{A} \cap \overline{B} \Rightarrow x \in \overline{A} \wedge x \in \overline{B} \Rightarrow x \notin A \wedge x \notin B \Rightarrow \neg(x \in A) \wedge \neg(x \in B) \Rightarrow \neg(x \in A \vee x \in B) \Rightarrow \neg(x \in A \cup B) \Rightarrow x \notin A \cup B \Rightarrow x \in \overline{A \cup B}$. Therefore, $\overline{A} \cap \overline{B} \subseteq \overline{A \cup B}$.

Proof 2. Instead of proving both ways use the identities:

$$\begin{aligned} \overline{A \cup B} &= \{x | x \notin A \cup B\} = \{x | \neg(x \in A \cup B)\} \\ &= \{x | \neg(x \in A \vee x \in B)\} = \{x | \neg(x \in A) \wedge \neg(x \in B)\} \\ &= \{x | x \notin A \wedge x \notin B\} = \{x | x \in \overline{A} \wedge x \in \overline{B}\} \\ &= \{x | x \in \overline{A} \cap \overline{B}\} = \overline{A} \cap \overline{B} \end{aligned}$$

Proof 3. In a membership table (like a truth table), 1s and 0s are used to specify if an arbitrary element x is in a set expression or not. The rows in the table are all possible membership combinations of x. If the columns for the two expressions are identical then the expressions are equal.

A	B	$\overline{A}$	$\overline{B}$	$A \cup B$	$\overline{A \cup B}$	$\overline{A} \cap \overline{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Basic Structures: Functions, Lecture 7

Functions *Reading: Rosen 2.3* This should be just a review of high school material.

- A **function** $f : A \to B$ is a mapping of each element in set A to some element in set B. A is called the **domain** of f and B is called the **codomain** of f.

  The set $\{f(a)|a \in A\}$ is a subset of B and is called the **range** of f.

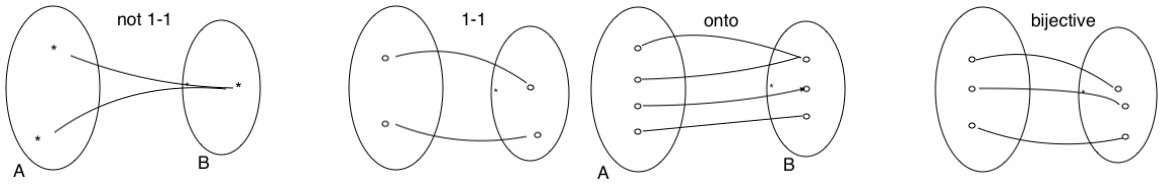
  To generalize to multi-variables, function $f : A_1 \times A_2 \times \cdots \times A_n \to B$ is a mapping of each ordered n-tuple to some element in B.

- Function $f : A \to B$ is **injective** or **one-to-one** or **1-1** iff
  $f(a) = f(b) \to a = b, \ \ \forall a, b \in A$
  That is, there are no different values in A that are mapped to the same value in B.

  Function f is **surjective** or **onto** iff the range of f coincides with its codomain
  $\forall b \in B \ \exists a \in A : b = f(a)$.

  Function f is **bijective** if it is both injective and surjective.

- For a 1-1 function $f : A \to B$, its inverse is defined to be
  $f^{-1} : B \to A$, where $f^{-1}(b) = a$ when $f(a) = b$.



- $f : A \to B$ and $g : B \to C$. Then define composition function as:
  $g \circ f : A \to C$, with $g \circ f(a) = g(f(a))$

  Note that if f is bijective (i.e., the inverse exists) then
  $f \circ f^{-1} = \mathbf{1}_{B \to B}$ and $f^{-1} \circ f = \mathbf{1}_{A \to A}$

- The definitions of functions to be **increasing** or **non-decreasing**, **strictly increasing**, **decreasing** or **non-increasing**, and **strictly decreasing**.

Here is a summary of some common functions used in computer science:

- Ceilings and floors: $\lceil x \rceil$ and $\lfloor x \rfloor$

  $\lceil \ \rceil : \mathbb{R} \to \mathbb{Z} : \lceil x \rceil =$ smallest int $\geq x$
  $\lfloor \ \rfloor : \mathbb{R} \to \mathbb{Z} : \lfloor x \rfloor =$ largest int $\leq x$

  Examples: $\lfloor 1/2 \rfloor = 0$, $\lfloor -1/2 \rfloor = -1$, $\lceil 3.1 \rceil = 4$, $\lceil 8 \rceil = 8$

- Modular arithmetic: $a \bmod n = a - \lfloor a/n \rfloor n$.

  $(a \equiv b \bmod n)$ iff $(a \bmod n = b \bmod n)$

  Examples: 5 mod 3 = 2, 6 mod 3 = 0, 9 mod 2 = 1
  Note -3 mod 2 = $-3 - \lfloor -3/2 \rfloor 2 = -3 - (-2)2 = +1$
  Similarly -2 mod 2 = 0, and -1 mod 2 = 1, mod(-8, 5) = 2, mod(-8,-5) = -3, etc

- Polynomials: $p(n) = \sum_{i=0}^{d} a_i n^i$. (Note: Coefficients a_i and degree d are constants.)
  Ex: $2x^2 + 2x + 1$, or $x^5 + x^2$

- Exponentials: $a^0 = 1$, $a^{-1} = \frac{1}{a}$, $a^m \cdot a^n = a^{m+n}$, $a^m / a^n = a^{m-n}$.

  $e = 2.71828\ldots$, the base of the natural logarithm

  $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ (see Taylor series in Calculus)

  $\lim_{n\to\infty} (1 + \frac{x}{n})^n = e^x$ (continuous compound interest)

  $\frac{de^x}{dx} = e^x$

- Logarithms: if $x \neq 0$, $\log_a x = l \in \mathbb{R} : a^l = x$ or simply $a^{\log_a x} = x$
  In CS we denote: $\log n = \log_2 n$ and $\ln x = \log_e x$
  Properties

  $\log(ab) = \log a + \log b$, $\log(\frac{a}{b}) = \log a - \log b$.

  $\log_a b^n = n \log_a b \neq (\log_a b)^n$, $a^{\log_a n} = n$

  $\log_a b = \frac{\log_c b}{\log_c a}$ (change of base). Proof: $c^{\log_c b} = b = a^{\log_a b} \rightarrow \log_c(c^{\log_c b}) = \log_c(a^{\log_a b}) \rightarrow \log_c b \log_c c = \log_a b \log_c a \rightarrow$QED

  $a^{\log_c b} = b^{\log_c a}$.

  $ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$

- Factorials: $n! = n \cdot (n-1) \cdots 2 \cdot 1 = \prod_{i=1}^{n} i$

  $n! = n \cdot (n-1)!$, $0! = 1$.

  Stirling's approximation: $n! = \sqrt{2\pi n}(\frac{n}{e})^n (1 + \Theta(\frac{1}{n}))$. (Note: Θ means having the same order of magnitude.) The following approximation also holds:
  $n! = \sqrt{2\pi n}(\frac{n}{e})^n e^{\alpha_n}$, where $\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$.

  $\log n! = \sum_{i=1}^{n} \log i \leq n \log n$. We'll see later it is $\Theta(n \log n)$

- $\log^* n = \min\{i \geq 0 : \log^{(i)} n \leq 1\}$, i.e., # of times to apply log to get below 1. Very slowly growing function! $\log^* 2 = 1$, $\log^* 4 = 2$, $\log^* 5 = 3$, $\log^* 2^4 = 3$, $\log^* 17 = 4, \ldots, \log^* 2^{16} = \log^* 65536 = 4$, $\log^* 65537 = 5, \ldots, \log^* 2^{65536} = 5, \ldots$

Basic Structures: Sequences/Sums, Lecture 8

Sequences and sums *Reading: Rosen 2.4*

A **sequence** is a discrete structure used to represent an ordered list. It can be defined as a function $s : \mathbb{N} \to S$, for some set S.

Examples:

- $a_1 =$ employee #1, ..., a_i=employee #i
- Think "array" but not necessarily finite
- $\{h_n\}$, where $h_n = \frac{1}{n}$, is the **harmonic sequence** $1, \frac{1}{2}, \frac{1}{3}, \cdots$.
- In some cases, a sequence may also start from $n = 0$.
- **Arithmetic progression**: $a, a+d, a+2d, \cdots, a+nd, \cdots$
- **Geometric progression**: $a, ar, ar^2, \cdots, ar^n, \cdots$
- Arith: $1, 2, 3, 4 \ldots$, another: 2,4,6,8,... another: 5,10,15,20,...
- Geom: $1, 2, 4, 8, 16, \ldots$
- Geom: $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ $(r = 1/2)$
- Geom: 1, 0.8, 0.64, 0.512, 0.4096, ... $(r = 0.8)$
- Perfect squares: $1, 4, 9, 16, 25, \ldots$
- Fibonacci: $1, 1, 2, 3, 5, 8, 13, \ldots$ or $f_n = f_{n-1} + f_{n-2}$ with $f_0 = 1, f_1 = 1$.

We want to capture the way something increases or decreases at discrete steps.
Often we want the total amount of effort, cost, etc
A **summation** is a sum of some terms. We use the Greek letter sigma, Σ,
$a_m + a_{m+1} + \cdots + a_n$ can be written as $\sum_{j=m}^{n} a_j$, $\sum_{k=m}^{n} a_k$, or $\sum_{m \le i \le n} a_i$.

Also, $\sum_{i=1}^{\infty} a_i = \lim_{n\to\infty} \sum_{i=1}^{n} a_i$.

The law of linearity: $\sum_{i=1}^{n} (ax_i + by_i) = a\sum_{i=1}^{n} x_i + b\sum_{i=1}^{n} y_i$.

Double summation: $\sum_{i=1}^{m} \sum_{j=1}^{n} (2i + 3^j)$. This is like two nested for-loops.

Factors that do not depend on the summation index can be factored out:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} a_i b_j = \sum_{i=1}^{n} (a_i b_1 + \ldots + a_i b_m) = \sum_{i=1}^{n} a_i (b_1 + \ldots + b_m) = \sum_{i=1}^{n} a_i \sum_{j=1}^{m} b_j$$

$$\text{Ex}: \quad \sum_{i=1}^{m}\sum_{j=1}^{n}(2i+3^j) = \sum_{i=1}^{m}(\sum_{j=1}^{n}2i+\sum_{j=1}^{n}3^j) = 2\sum_{i=1}^{m}i\sum_{j=1}^{n}1+\sum_{i=1}^{m}\sum_{j=1}^{n}3^j$$
$$= 2n\sum_{i=1}^{m}i+m\sum_{j=1}^{n}3^j = \text{2n (sum of arithm) + m *(sum of geom)}$$

A sequence can have **two indices** (or more)
We can visualize 2 indices (i.e., two nested sums) as a 2D array/matrix. Ex: $a_{ij} = (3i)^j$:

	$j=1$	$j=2$	$j=3$	$j=4$	$\ldots$
$i=1$	3	9	27	81	$\ldots$
$i=2$	6	36	216	1296	$\ldots$
$i=3$	9	81	$\ldots$		
$\vdots$	$\vdots$				

Double summation over two indices is like two nested for-loops $\sum_{i=1}^{m}\sum_{j=1}^{n}a_{ij}$

We can sum over these two indices in two different ways as long as the inner loop variables do not depend on the outer loop:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}a_{ij} = \sum_{j=1}^{m}\sum_{i=1}^{n}a_{ij}$$

Some usefully summations:

- Arithmetic series: $\sum_{k=1}^{n}k = 1+2+\cdots+n = \frac{1}{2}n(n+1)$ (recall baby Gauss).
  $\sum_{k=1}^{n}kd = d\sum_{k=1}^{n}k = \frac{1}{2}dn(n+1)$.
  $\sum_{k=0}^{n}(a+kd) = \sum_{k=0}^{n}a+\sum_{k=0}^{n}kd = a(n+1)+\frac{1}{2}dn(n+1) = \frac{1}{2}(n+1)(2a+nd)$.

- Geometric series: $\sum_{i=0}^{n}r^i = 1+r+r^2+\cdots+r^n = \frac{r^{n+1}-1}{r-1}$ if $r \neq 1$, but $\sum_{i=0}^{n}r^i = n+1$, if $(r=1)$.
  Proof: Let $S = \sum_{i=0}^{n}r^i$. Then $rS = r\sum_{i=0}^{n}r^i = \sum_{i=0}^{n}r^{i+1} = \sum_{i=1}^{n+1}r^i = \sum_{i=0}^{n}r^i + r^{n+1} - 1 = S + r^{n+1} - 1$. So $rS = S + r^{n+1} - 1$. Therefore, $S = \frac{r^{n+1}-1}{r-1}$.

- Sum of squares: $\sum_{j=1}^{n}j^2 = 1^2+2^2+3^2+\cdots+n^2 = \frac{1}{6}n(n+1)(2n+1)$.

- Sum of cubes: $\sum_{j=1}^{n}j^3 = 1^3+2^3+3^3+\cdots+n^3 = \frac{1}{4}n^2(n+1)^2$.
  Proof of squares: Let $S = \sum_{j=1}^{n}j^2$.

– First, let's replace j^2 with an arithmetic summation. Let a,d the parameters of the arithmetic series. Then $\sum_{i=1}^{j}(a+id) = aj + dj(j+1)/2 = j^2$. Choosing $d=2$ gives $a=-1$, and thus $\sum_{i=1}^{j}(2i-1) = j^2$.

– Thus,
$S = \sum_{j=1}^{n} j^2 = \sum_{j=1}^{n}\sum_{i=1}^{j}(2i-1) \stackrel{*}{=} \sum_{j=1}^{n}(2j-1)(n-j+1)$
$= \sum_{j=1}^{n}(2jn-2j^2+2j-n+j-1) = -2S+\sum_{j=1}^{n}(2jn+3j)-n(n+1)$
$= -2S+(2n+3)n(n+1)/2-n(n+1) \rightarrow 3S = n(n+1)(2n+1)/2.$

$\stackrel{*}{=}$ How to analyze the double summation:
$\sum_{j=1}^{n}\sum_{i=1}^{j}(2i-1) =$

$(j=1)$	1	+										
$(j=2)$	1	+	3	+								
$(j=3)$	1	+	3	+	5	+						
	...											
$(j=n)$	1	+	3	+	5	+	...	+	2n-1			
	n	+	$3(n-1)$	+	$5(n-2)$	+	...	+	$(2n-3)2$	+	$(2n-1)1$	=

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

- Differentiating series: $\sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2}$ for $|x|<1$.

  Proof: Consider the sum $\sum_{k=0}^{n} kx^{k-1}$ first. Using differentiation, we get $\sum_{k=0}^{n} kx^{k-1} = \sum_{k=0}^{n}(x^k)' = (\sum_{k=0}^{n} x^k)' = (\frac{1-x^{n+1}}{1-x})' = \frac{-(n+1)x^n(1-x)-(1-x^{n+1})(-1)}{(1-x)^2} = \frac{nx^{n+1}-(n+1)x^n+1}{(1-x)^2}$.

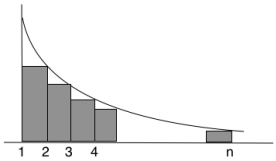
  So $\sum_{k=0}^{\infty} kx^{k-1} = \lim_{n\to\infty}\sum_{k=0}^{n} kx^{k-1} = \lim_{n\to\infty}\frac{nx^{n+1}-(n+1)x^n+1}{(1-x)^2} = \frac{1}{(1-x)^2}$.

  Alternative 1: $\sum_{k=0}^{\infty} kx^{k-1} = \sum_{k=0}^{\infty}(x^k)' = (\sum_{k=0}^{\infty} x^k)' = (1/(1-x))' = 1/(1-x)^2$

  Alternative 2: $S = \sum_{k=1}^{\infty} kx^{k-1} \rightarrow S - xS = 1+2x+3x^2\ldots - x - 2x^2 - 2x^3\ldots =$
  $1+x+x^2+\ldots = \frac{1}{1-x} \rightarrow S = 1/(1-x)^2$

- Harmonic series: The nth harmonic number is $H_n = \sum_{i=1}^{n}\frac{1}{i} < \ln n + 1$

  Proof: $H_n = \sum_{i=1}^{n}\frac{1}{i} = 1+\sum_{i=2}^{n}\frac{1}{i} < 1+\int_1^n \frac{1}{x}dx = 1+\ln n$



- Telescoping series: For any sequence $a_0, a_1, \ldots, a_n$, $\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0$.
  For example, using telescoping, we can show that $\sum_{k=1}^{n-1}\frac{1}{k(k+1)} = 1-\frac{1}{n}$.

  Proof: $S = \sum_{k=1}^{n-1}(\frac{1}{k}-\frac{1}{k+1}) = -\sum_{k=1}^{n-1}(\frac{1}{k+1}-\frac{1}{k})$. Let $a_k = \frac{1}{k+1}$. Then,
  $-\sum_{k=1}^{n-1}(a_k - a_{k-1}) = a_0 - a_{n-1} = 1-1/n$.

Growth rates of functions, Lecture 9

Reading: Rosen 3.2

We need a measure to compare the efficiency of algorithms (for a given criterion, time, space, energy, etc). We measure flops, ops, bytes, Joules, etc. The final functions are increasing (non-decreasing) with problem size.

We need to get a "feel" of how fast these functions grow as their variables grow.

- **Big-O notation** is used to compare the growth rates of two increasing functions, $f(x)$ and $g(x)$. We say that $f(x)$ is big-O of $g(x)$, also written as $f(x) = O(g(x))$ or $f(x) \in O(g(x))$, iff

$$\exists c, k \text{ constants} : f(x) \leq cg(x),\ \forall x \geq k$$

  $f(x) = O(g(x))$ implies that the growth rate of $f(x)$ is less than or equal to the growth rate of $g(x)$.

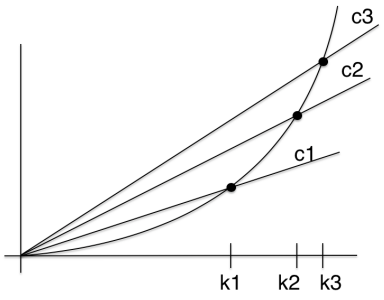
- Example (1) $x^2 + 2x + 1 = O(x^2)$.

  For $c = 2 \Rightarrow x^2 + 2x + 1 \leq 2x^2 \Leftrightarrow 2x + 1 \leq x^2$ which is true $\forall x \geq 3 \equiv k$. Note for a different c, say $c = 10$, we would get a different k, e.g, $\forall x \geq 1$.

- Example (2) $100x^2 = O(x^3)$.

  $100x^2 \leq cx^3 \Leftrightarrow x \geq 100/c$. Pick a $c > 1$, then holds for $x \geq k = 100/c$.

- Different c_i imply different k_i



- Example $2x = O(3x)$, pick $c = 1, k = 1$

- Example $3x = O(2x)$, pick $c = 2, k = 1$ (they grow at the same rate)

- Transitivity: If $f(x) = O(g(x))$ and $g(x) = O(h(x))$, then $f(x) = O(h(x))$.

- If $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ is a polynomial, then $f(x) = O(x^n)$, i.e., a polynomial grows at the rate of its highest degree. Lower-order terms can be ignored. Constant factors are absorbed in the constant c in the big-O definition.

  Intuitively: $a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \leq \max_{0 \leq i \leq n-1}(x^{n-1} + x^{n-2} + \ldots + 1) = \max(a_i)\frac{x^n - 1}{x-1} \leq \max(a_i)x^n, \forall x > 1$. Then $f(x) \leq (a_n + \max(a_i))x^n$, $\forall x > 1$.

- Ex. (1) $1+2+\cdots+n \leq n+n+\cdots+n = n^2$ thus $\sum_{i=0}^{n} i = O(n^2)$, or $\sum_i i \leq \sum_i n = n\sum_i = n^2$
- Ex. (2) $n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n \leq n \cdot n \cdot n \cdot \cdots \cdot n = n^n$ thus $n! = O(n^n)$.
- Ex. (3) $n = O(2^n) \rightarrow \log n = O(n)$. Also, $n = O(b^n) \rightarrow \log_b n = O(n)$.
- Ex. (4) $\log n^m = m \log n < O(n)$
- Ex. (5) $n^m = O(2^n)$ Note that $\log n = O(n) \rightarrow \forall m$ const, $m \log n = O(n) \rightarrow 2^{m \log n} = O(2^n) \rightarrow 2^{m \log n} = 2^{\log n^m} \rightarrow n^m = O(2^n)$.
- Ex. (6) $(\log n)^m = O(n)$ (using the L'Hospital rule)
- Common functions ordered by increasing growth rates: $1, \log n, n, n \log n, n^2, 2^n, n!$. Rule of Thumb: **Const** < **Polylog** < **Polynomial** < **Exponential**

  Example: Order the following functions by increasing growth rate.

  n^2, $(\log n)^m$, n^3, $(logn)^n$, $n^2 \log n$, 2^n, n^n, $n^{\log n}$, 2^{2^n}, $n!$, $2^{2^{n+1}}$.

  Ans: $(\log n)^m$, n^2, $n^2 \log n$, n^3, $n^{\log n}$, 2^n, $(logn)^n$, $n!$, n^n, $2^{2^n} \neq 2^{2^{n+1}} = (2^{2^n})^2$
- **Big-Ω**: We say that $f(x) = \Omega(g(x))$ iff $\exists c, k$ constants : $f(x) \geq cg(x)\ \forall x \geq k$. Also, $f(x) = \Omega(g(x))$ implies that the growth rate of $f(x)$ is greater than or equal to the growth rate of $g(x)$. In addition, $f(x) = \Omega(g(x))$ iff $g(x) = O(f(x))$.
- **Big-Θ**: $f(x) = \Theta(g(x))$ iff $\exists c_1, c_2, k$ constants : $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for $n \geq k$. Also, $f(x) = \Theta(g(x))$ implies that the growth rate of $f(x)$ is equal to the growth rate of $g(x)$. In addition, $f(x) = \Theta(g(x))$ iff $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.
- **Little-o**: $f(n) = o(g(n))$ iff $\forall c\ \exists k$ such that $f(n) < cg(n)$ for $n \geq k$. An alternative definition for $f(n) = o(g(n))$ is $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- **Little-ω**: $f(n) = \omega(g(n))$ if $\forall c\ \exists k$ such that $f(n) > cg(n)$ for $n \geq k$. An alternative definition for $f(n) = \omega(g(n))$ is $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

exp, log are monotonic so if $(f(x) < cg(x) \rightarrow e^{f(x)} < e^{cg(x)})$.
$(f = o(g) \rightarrow e^f = O(e^g))$
$(f = O(g) \rightarrow \log f = O(\log g))$
Exponentiation accentuates differences. So if $f = \Theta(g)$, e.g., $2x = \Theta(x)$, yet $e^{2x} = (e^x)^2 \neq O(e^x)$. Conversely, log may compress differences down to Θ.

Algorithms, Lecture 10

Reading: Rosen 3.1,3.3

An **algorithm** is a *finite* set of *precise* instructions for performing a computing *task* or for solving a problem.

Some components of an algorithm:

I/O: input, output (solution/answer)

definiteness (well-defined steps)

correctness (provably)

generality (working for all possible inputs of a certain category)

finiteness (no infinite loops

effectiveness (performable steps),

time, space, power, energy, real-time, QoS

focus on time and space as a function of problem (input) size. E.g., size of the database to search, number of cities to consider in the traveling salesperson, etc

We use the theory we learned to sum a series of numbers of operations and express them in terms of asymptotic notation (big-O).

An algorithm can be described by words or by pseudocode, which is the combination of programming languages, mathematical notation, and English.

Example 1: Find the maximum number in a list of integers.

7 5 2 8 1

```
1. Set the current max to be the first number in the list.
2. Compare the next integer in the list with the current max,
   and if it is larger, then set it to be the new current max.
3. Repeat step 2 until the numbers in the list have all been compared.
4. Output the current max as the maximum number of the list.
```

Note that if max starts as something large (e.g., 1000) then if A(i) = 1001 would fail. To avoid guessing an upper bound we set it to the 1st element.

Pseudocode:

Algorithm Find_Max ($a_1, a_2, \ldots, a_n$: integers)

$max \leftarrow a_1$

for $i \leftarrow 2$ to n

if $max < a_i$ then $max \leftarrow a_i$

return max

Corrrectness? What if n=0 (empty list)? Alg returns max (undefined, random?). We want to guard against such cases and require a pre-condition that $n > 0$ outside the algorithm, or raise an exception inside the algorithm.

Finiteness/Effectiveness. Find_max executes at most n times (which we assume as input it is finite).
Each step involves a comparison (constant time in Hardware) and an assigment (constant time also)
Its execution time in the worst case is n comparisons + n assigments = $nc_1 + nc_2 = n(c_1 + c_2)$, i.e., $O(n)$ which shows the growth rate independently of the architectural paramers (c_1, c_2)

Example 2: Search for integer x among an *unordered* list of n integers.
Algorithm Linear_Search(x: integer; $a_1, a_2, \ldots, a_n$: integers)
$i \leftarrow 1$
while $i \leq n$ and $x \neq a_i$
  $i \leftarrow i + 1$
if $i \leq n$ then $location \leftarrow i$
else $location \leftarrow 0$
Correctness: Incementation of i occurs when the condition is true, i.e., $i \leq n \wedge x \neq a_i$.
While exits when condition is false or: $i > n \vee x == a_i$ (de Morgan) which specifies what to expect on exit.
Therefore, if $i \leq n$ we exited because we found $x == a_i$.
if $i > n$, it means we visited all elelents and we didn't find x, thus $location = 0$.
Effectiveness: Obviously finite and $O(n)$ worst case time

Example 2: Search for integer x among an *ordered* list of n integers.
Here we can do better:
Algorithm Binary_Search(x: integer; $a_1, a_2, \ldots, a_n$: increasing integers)
$i \leftarrow 1$
$j \leftarrow n$
$location \leftarrow 0$
while $i \leq j$ and $location = 0$
  $m \leftarrow \lfloor (i + j)/2 \rfloor$
  if $x > a_m$ then $i \leftarrow m + 1$
  else if $x < a_m$ then $j \leftarrow m$
    else $location \leftarrow m$
return $location$
E.g. Binary search

```
1  2  3  4  5  6  7  8  9  10 11  12  13  14  15  16
i                     m                             J
i        m            j
            i  m      j
                  i/m j  done
```

Effectiveness/time Complexity

Every step we throw out about 1/2 of the elements

step 1	after a const # of ops (less than 10) the elements remaining:	$\frac{n}{2} = n/2^1$
step 2		$\frac{n}{4} = n/2^2$
step 3		$\frac{n}{8} = n/2^3$
$\vdots$		$\vdots$
step k		$n/2^k$

We stop when $n/2^k = 1 \Leftrightarrow log_2 n = log_2 2^k = k$. Thus $O(\log n)$ time.
E.g., 10^9 elements will be searched in $9 \log 10 \approx 27$ steps.

Complexity of algorithms

Reading: Rosen 3.3

The **computational complexity** of an algorithm includes **time complexity** and **space complexity**. Unless otherwise stated, we focus on the time complexity of algorithms.
The time complexity of an algorithm is defined as the number of basic steps the algorithm goes through from input to output.
It is usually represented as a function $T(n)$ of the size of input n and simplified by O and Θ, which emphasizes the asymptotic performance of algorithms for large input sizes and makes it easy to compare complexity of different algorithms.
The definition of time complexity makes the analysis of algorithms input-value independent, programming independent, language independent, and machine independent.
We usually examine **worst-case** complexity **Average case** complexity is also popular (involving probability and averaging (expectation value) over all inputs).
Definition:
I_n: Any instance of size n;
$t(I_n)$: Time (# of basic steps) spent on I_n by the algorithm;
$T(n)$: Worst-case time complexity of any instance of size n, $T(n) = \max_{\forall I_n}\{t(I_n)\}$.
$A(n)$: Average-case time complexity over all possible instances is $A(n) = \mathbb{E}(t(I_n))$, i.e., the expectation value of $t(I_n)$.
Beyond these, a very difficult analysis is the **lower bound analysis**. This seeks to find a lower bound function $f(n)$ such that any algorithm for solving a particular problem cannot have worst case time less than $f(n)$, or in other words $T(n) = \Omega(f(n))$ for any algorithm solving this problem.

Example of lower bound for the problem of finding x in a list of n unordered numbers.
Say we have a magic search algorithm that performs less than n comparisons. Then it must leave out some element, say a_j. Then if $x = a_j$, our magic algorithm cannot find it.
This is an example of the **Adversary Technique** used to find lower bounds: Always picks the worst possible input for an algorithm.

Algorithms, Lecture 11

Worst-case time complexity:

- Find Max and Linear Search are worst case linear-time algorithms, i.e., $\Theta(n)$
- Binary Search redux: For simplicity assume $n = 2^k$. Each step the list is reduced by half. Thus, $T(n) = 1 + T(n/2) = 1 + 1 + T(n/2^2) = 1 + 1 + 1 + T(n/2^3) = \ldots = k + T(n/2^k) = k + 1 = \log n + 1 = \Theta(\log n)$.
- Sorting into increasing order.

  Algorithm **Bubble_Sort**($a_1, a_2, \ldots, a_n$: reals with $n \geq 2$)
  for $i \leftarrow 1$ to $n-1$

  for $j \leftarrow 1$ to $n-i$

  if $a_j > a_{j+1}$ then swap a_j and a_{j+1}

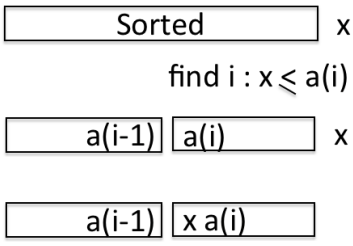
  Example:
  8 5 2 4 3 1 $\rightarrow$ 5 2 4 3 1 $\underline{8} \rightarrow$ 2 4 3 1 $\underline{5\ 8} \rightarrow$ 2 3 1 $\underline{4\ 5\ 8} \rightarrow$ 2 1 $\underline{3\ 4\ 5\ 8} \rightarrow$ $\underline{1\ 2\ 3\ 4\ 5\ 8}$

  Time Complexity: First loop executes n-1 times. Everything inside it executes from 1 to $n-i$. The if statement takes a constant (2-3) operations. Thus

$$ops = \sum_{i=1}^{n-1} \sum_{j=1}^{n-i} 3 = 3 \sum_{i=1}^{n-1} (n-i) = 3 \sum_{i=1}^{n-1} n - 3 \sum_{i=1}^{n-1} i$$

$$= 3(n(n-1) - n(n-1)/2) = \frac{3}{2} n(n-1) = \Theta(n^2)$$

  Algorithm **Insertion_Sort**($a_1, a_2, \ldots, a_n$: reals with $n \geq 2$)
  for $j \leftarrow 2$ to n //insert a_j into the sorted $a_1, \ldots, a_{j-1}$
  $key \leftarrow a_j$
  $i \leftarrow j-1$
  while $i > 0$ and $a_i > key$
  $a_{i+1} \leftarrow a_i$
  $i \leftarrow i-1$
  $a_{i+1} \leftarrow key$



  Time complexity of Insertion Sort: To insert a_j into the sorted $a_1, \ldots, a_{j-1}$, the algorithm will make at most $j-1$ comparisons and $j-1$ shifts. So the overall time complexity, which is dominated by the number of comparisons and the number of shifts, is at most $2\sum_{j=2}^{n}(j-1) = \Theta(n^2)$. The best case is $\Theta(N)$ if i stops at $j-1$. More interestingly, if elements are uniformly distributed, the average number of iterations of the while loop should be $(j-1)/2$. Thus, the average case analysis gives half the time of the worst-case (although still quadratic).

- Complexity of nested loops:
  A loop runs through a certain number of times which corresponds to summing whatever ops each iteration performs. So, a loop

```
for i=1 to n
   MethodA(i)
   MethodB(i)
```

  takes time $\sum_{i=1}^{n}(\text{Ops(Method}(i))+\text{Ops(MethodB}(i)))$

  Each of these methods could be a nested loop too. Every time we see a loop we nest a summation:

```
for i=1 to n
   for j = 1 to i
      Method(j)
```

  takes time $\sum_{i=1}^{n}\sum_{j=1}^{i}\text{Ops(Method}(j))$

  While loops need to be examined carefully to see how many times they execute and how their variable changes.

```
for i=1 to n
   j=i
   while j >= 1
      for l=1:j
          A(l)++
      j = j/2
```

  takes time: $\sum_{i=1}^{n}\sum_{k=0}^{\log i}\sum_{1}^{i/2^k}1 =$
  $\sum_{i=1}^{n}\sum_{k=0}^{\log i} i/2^k = \sum_{i=1}^{n} i\sum_{k=0}^{\log i}(1/2)^k =$
  $\sum_{i=1}^{n} i\frac{(1/2)^{\log i+1}-1}{1/2-1} = \sum_{i=1}^{n} 2i(1-(1/2^{\log(2i)})) =$
  $\sum_{i=1}^{n} 2i(1-1/(2i)) = \sum_{i=1}^{n} 2i-1 = n(n+1)-n = n^2$

- Commonly appearing time complexity functions:
  Constant complexity: $\Theta(1)$
  Logarithmic complexity: $\Theta(\log n)$
  Linear complexity: $\Theta(n)$
  $n\log n$ complexity: $\Theta(n\log n)$
  Polynomial complexity: $\Theta(n^d)$ for positive d
  Exponential complexity: $\Theta(b^n)$ for $b > 1$
  Factorial complexity: $\Theta(n!)$

Induction and its extensions, Lecture 12

Reading: Rosen 4.1

Principle of induction: $(P(1) \wedge \forall k(P(k) \rightarrow P(k+1))) \rightarrow \forall n P(n)$.

Notice the base case, and the inductive hypothesis $\rightarrow$ inductive step.

Example 1: Sum: $\sum_{j=0}^{n} r^j = \frac{r^{n+1}-1}{r-1}$ for $r \neq 1$.

Proof: Basis step: The equation holds true when $n = 0$ since $\sum_{j=0}^{0} r^j = 1 = \frac{r^{0+1}-1}{r-1}$.

Inductive Hypothesis: Assume $\sum_{j=0}^{k} r^j = \frac{r^{k+1}-1}{r-1}$.

Inductive Step: Then $\sum_{j=0}^{k+1} r^j = \sum_{j=0}^{k} r^j + r^{k+1} = \frac{r^{k+1}-1}{r-1} + r^{k+1} = \frac{r^{k+2}-1}{r-1}$.

Example 2: If $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the sum of the harmonic series, $H_{2^n} \geq 1 + \frac{n}{2}$, $\forall n \geq 0$.

Proof: Basis step: The inequality holds for $n = 0$ since $H_{2^0} = H_1 = 1 = 1 + \frac{0}{2}$.

I.H.: Assume $H_{2^k} \geq 1 + \frac{k}{2}$.

I.S.: Since $2^{k+1} = 2^k + 2^k$, we can split $H_{2^{k+1}}$ into H_{2^k} and the remaining 2^k terms: $H_{2^{k+1}} = H_{2^k} + (\frac{1}{2^k+1} + \frac{1}{2^k+2} + \cdots + \frac{1}{2^{k+1}}) \geq (1 + \frac{k}{2}) + (\frac{1}{2^k+1} + \frac{1}{2^k+2} + \cdots + \frac{1}{2^{k+1}}) \geq 1 + \frac{k}{2} + 2^k \cdot \frac{1}{2^{k+1}} = 1 + \frac{k+1}{2}$.

Example 3: Divisibility: $n^3 - n$ is divisible by 3 for integer $n \geq 1$.

Proof: Basis step: For $n = 1$, $n^3 - n = 0$, which is certainly divisible by 3.

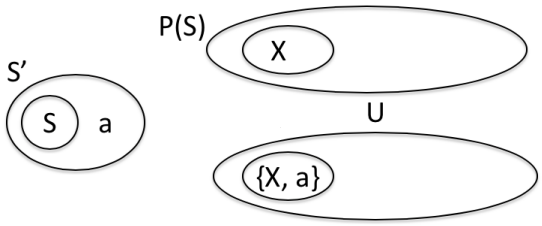
I.H.: Assume that $k^3 - k$ is divisible by 3. I.S.: Then $(k+1)^3 - (k+1) = (k^3 + 3k^2 + 3k + 1) - (k+1) = (k^3 - k) + (3k^2 + 3k) = (k^3 - k) + 3k(k+1)$, which is divisible by 3 since both terms are divisible by 3.

Example 4: Set: For any set S, if $|S| = n$, then $|2^S| = 2^n$.

Proof: Basis step: For $n = 0$, $S = \emptyset$. So $2^S = \{\emptyset\}$. Therfore, $|2^S| = 1 = 2^0$.

I.H.: Assume that if $|S| = k$ then $|2^S| = 2^k$.

I.S.: Consider that $S' = S \cup \{a\}$ with $|S'| = k+1$. Then $2^{S'} = 2^S \cup \{X \cup \{a\} | X \in 2^S\}$. So $|2^{S'}| = |2^S| + |2^S| = 2 \cdot |2^S| = 2 \cdot 2^k = 2^{k+1}$.



Strong induction

Reading: Rosen 4.2

Principle of strong induction: To prove that $P(n)$ is true for integer $n \geq 1$

$$(P(1) \wedge (\forall k \geq 1, P(1) \wedge P(2) \wedge \cdots \wedge P(k) \rightarrow P(k+1))) \rightarrow \forall n P(n)$$

Basis step: Verify that $P(1)$ is true;

Inductive Hypothesis: Assume $(P(1) \wedge P(2) \wedge \cdots \wedge P(k))$

Inductive Step: Prove: $P(k+1)$ for integer $k \geq 1$.

Example: Any $n > 1$ can be written as a product of one or more primes.
Base case: $n = 2$, 2 is a prime.
I.H.: Assume $\forall j \leq k$, j is a product of primes.
I.S.: Consider $k+1$. (1): $k+1$ is prime, done
(2): $k+1$ is composite $\Rightarrow \exists a,b : 2 \leq a,b < k+1$ with $k+1 = ab \stackrel{I.H.}{=} \prod_i p_i \prod_j p_j = \prod_m p_m$.
Example: The remove-the-dot game.
There are 2 players, 2 rows of dots. The number of dots per row, n_1, n_2, may differ.
..... n_1
..... n_2
Rule 1: A player removes any positive number of dots from only 1 row at a time.
Rule 2: The player who removes the last dot wins.
Prove: If $n_1 = n_2 = n$, the player that plays 2nd can be guaranteed to win.
Base case: $n = 1$. First player must choose a row. Because there is only one dot, the player must remove it. Then the 2nd player simply removes the other row's dot and wins.
I.H.: Assume true for any number of dots $1 \leq j \leq k$, i.e., the 2nd player can choose a winning strategy.
I.S.: Let $k+1$ dots per row. 1st player removes $1 \leq r \leq k+1$ dots from (say) 1st row.
If $r = k+1$, the 2nd player can remove all dots from 2nd row and win.
If $r < k+1$, the 2nd player can remove r from the other row. Now both rows have $k+1-r \leq k$ dots, and by I.H. the 2nd player can choose a strategy to win.

Notice the above theorem provides a winning strategy for 1st player if $n_1 \neq n_2$. Then 1st player will remove $|n_1 - n_2|$ dots and make the rows equal. Then, she just keeps matching the moves of the other player as the theorem suggests.

A slightly stronger form of induction
$(P(b) \wedge P(b+1) \wedge \ldots \wedge P(b+j) \wedge$
$(\forall k > b+j \wedge \forall i, b \leq i \leq k, P(i) \rightarrow P(k+1))) \rightarrow \forall n P(n)$
i.e., many base cases, and a strong assumption that can take us from any $i < k$ to $k+1$.
Note that $k > b+j$ so that we do not use the rule in our base cases.

Example: Any postage of 12 cents or more can be formed using 4cent and 5cent stamps.
Proof. Base: 12=3*4, 13=5+2*4, 14=2*5+4, 15=3*5.
I.H.: Assume true for all $12 \leq j \leq k$, with $k > 15$.
I.S.: Consider postage $k+1 = 4+k+1-4 = 4+(k-3)$. By I.H., $k-3 = a*4+b*5$.
Then $k+1 = (a+1)*4+b*5$.
Example: Let $a_0 = 0, a_1 = 4, a_n = 6a_{n-1} - 5a_{n-2}$. Prove $a_n = 5^n - 1$. Base: $n = 0,1$ holds. I.H.: $a_i = 5^i - 1, \forall i \leq k$. I.S.: $a_{+1} = 6a_k - 5a_{k-1} \stackrel{I.H.}{=} 6(5^k - 1) - 5(5^{k-1} - 1) = 6(5^k) - 6 - 5^k + 5 = 5*5^k - 1 = 5^{k+1} - 1$. Simple induction is not sufficient here.

Recursive definitions, Lecture 13

Reading: Rosen 4.3

- Recursively defined functions:

  We use two steps to define a function $f : N \to N$:

  Basis step: Specify the value of the function at some starting point, e.g., at 0 or 1,

  Recursive step: Give a rule for finding its value at an integer from its value at smaller integers.

  Example: $F(0) = 1$ and $F(n) = nF(n-1)$ for $n \geq 1$ (the $n!$)

  Example: $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$ (Fibanacci)

  Example: $T(1) = 1$ and $T(n) = 1 + T(\lfloor \frac{n}{2} \rfloor)$ for $n \geq 2$ (time for binary search)

  Example: $T(1) = 1$ and $T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n$ for $n \geq 2$ (time for fast sorting)

  Example: $f(0) = 1,\ f(n) = f(n-1) + r^n,\ n \geq 1$ (sum of the $n+1$ terms of the geometric series)

  Example: $f(2) = 1, f(n) = 1 + f(\log n), \forall n > 2$. What is this function?
  If $n = 2^k, f(n) = 1 + f(\log 2^k) = 1 + f(k)$. Similarly $f(2^{2^k}) = 1 + f(2^k) = 2 + f(k)$.
  E.g., $f(2^{16}) = 1 + f(16) = 1 + 1 + f(4) = 1 + 1 + 1 + f(2) = 4$. Then $f(2^{2^{16}}) = 5$.
  So $f = \log^*$, i.e., the number of times we must apply log to get to 1.

  For non-powers of 2 it needs broader definition: $f(n) = 1$, if $0 < n \leq 2$
  $f(n) = 1 + f(\log n)$ for $n > 2$

- Recursively defined sets:

  Example: Basis: $2 \in S$. Recursive step: if $x, y \in S$, then $x + y \in S$ (Positive even numbers).

  Example: The set Σ^* of all strings over the alphabet Σ can be defined as:

  Basis step: Empty string $\varepsilon \in \Sigma^*$ (i.e., $\varepsilon = \emptyset$)

  Recursive step: For any $w \in \Sigma^*$ and $a \in \Sigma$, $wa \in \Sigma^*$.

  E.g., $\Sigma = \{0,1\}.\Sigma^* = \{\emptyset, \{0\}, \{1\}, \{00\}, \{01\}, \{10\}, \{11\}, \{000\}, \ldots\}$

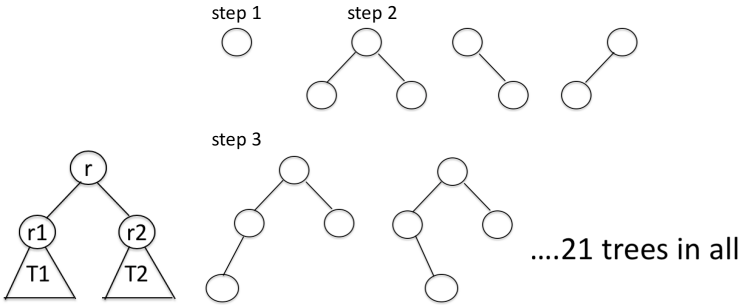
  A related function is the length of a string, defined recursively similarly to the above: $l(\varepsilon) = 0$ and $l(wx) = l(w) + 1, \forall w\Sigma^*,\ x \in \Sigma$

- Recursively defined structures:

  Example 5: The structure *Binary Tree* can be defined as

  Basis step: The empty tree (set) is a binary tree.

  Recursive step: For any binary trees T_1 with root r_1 and T_2 with root r_2, a new root r having T_1 as its left subtree and T_2 as its right subtree forms a binary tree.



  Structural induction is used to prove properties of recursive structures. To prove a property $S(X)$ of a recursively defined structure X, it is sufficient to prove that $S(X)$ is true for the basis structures of X and that $S(Y_1) \wedge S(Y_2) \wedge \cdots \wedge S(Y_k) \rightarrow S(X)$, where X is constructed from $Y_1, Y_2, \ldots, Y_k$.

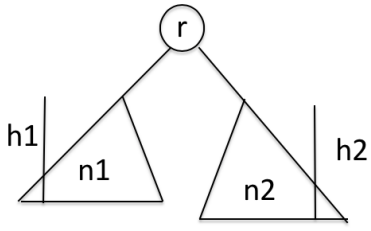
  Example: Let the height of a tree be the number of nodes on the longest path from the root to a leaf (note height of $\emptyset$ is 0). Prove that for any binary tree with n nodes and height h, $n \leq 2^h - 1$.

  Proof by inducting on height:

  Proof: Basis step: Consider the empty tree. Obviously $n = 0$ and $h = 0$, which satisfies the inequality.

  I.H.: Assume for any binary tree with height up to k, the inequality holds.

  I.S.: Consider a binary tree with height $k+1$.



  Assume that the left subtree has n_1 nodes with height h_1 and the right subtree has n_2 nodes with height h_2. Thus $k+1 = \max\{h_1, h_2\} + 1$.

  Since $h_1, h_2 \leq k$, then by the inductive hypothesis, $n_1 \leq 2^{h_1} - 1$ and $n_2 \leq 2^{h_2} - 1$. So $n = n_1 + n_2 + 1 \leq (2^{h_1} - 1) + (2^{h_2} - 1) + 1 \leq 2^{\max\{h_1, h_2\}+1} - 1 = 2^{k+1} - 1$.

  Structural Induction proof:

  Proof: Basis step: Consider the empty tree. Obviously $n = 0$ and $h = 0$, which satisfies the inequality.

I.H.: Assume for two binary trees T_1 and T_2 the inequality holds.

I.S.: We will prove that the inequality holds also for the tree built recursively from these two trees. This tree is built as a new root with T_1 and T_2 as its left and right subtrees.

Assume that T_1 has n_1 nodes with height h_1 and T_2 has n_2 nodes with height h_2. Thus, the height of the new tree is $h = \max\{h_1, h_2\} + 1$.

By I.H. $n_1 \leq 2^{h_1} - 1$ and $n_2 \leq 2^{h_2} - 1$. So $n = n_1 + n_2 + 1 \leq (2^{h_1} - 1) + (2^{h_2} - 1) + 1 \leq 2^{\max\{h_1,h_2\}+1} - 1 = 2^h - 1$. QED

Notice that the two proofs are almost identical. Structural induction simply obviates the need to induct on a particular variable, instead using the recursive definition as the step between I.H. and I.S.

Solving problems recursively, Lecture 14

Reading: Rosen 4.4 and 7.1

Think top down: How can I break the problem into a number of smaller problems from which I can built it through the given properties? Then, decide when to stop (basis). Examples:

• Computing $n!$:

```
factorial(n)
if n=0 return 1
else return n*factorial(n-1)
```

• Computing a^n linear time:

```
power(a, n)
if n=0 return 1
else return a*power(a,n-1)
```

Computing a^n in $\log n$ time:

```
power(a, n)
if n=0 return 1
else if n is even
         half=power(a,n/2)
         return half*half
     else return a*power(a, n-1)
```

• Binary search:

```
Binary_Search(i, j, x)
if i=j then compare x with a[i] and return T or F
m=(i+j)/2
if x=a[m] return T
else if x<a[m] return Binary_Search(i,m-1,x)
     else return Binary_Search(m+1,j,x)
```

• Merge sort

```
Merge_Sort(i,j)
if i<j
  m=(i+j)/2
  Merge_Sort(i,m)
  Merge_Sort(m+1,j)
  merge the two sorted sublists into one sorted list
```

Some problems that can be solved recursively

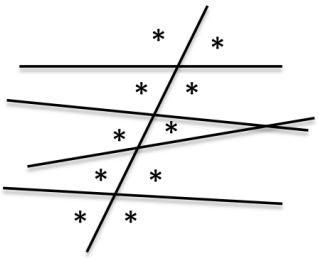
• What is the largest number of slices of pizza we can get with n straight cuts using a pizza knife? Or equivalently, what is the maximum number of regions created by n straight lines?

Let L_n be the answer. Then $L_0 = 1$ and $L_1 = 2$.

Assume now that I have already $n-1$ lines which create L_{n-1} regions.

To maximize the cuts, and thus the pieces, the new n-th line should cross all previous ones. Let the new line cross the previous ones in that order: $i_1, i_2, \ldots, i_{n-1}$.

The exterior regions before the new line crosses i_1 and after crossing i_{n-1} are separated into two regions. The $n-2$ interior regions defined between i_k and $i_{k+1}, 1 \leq k \leq n-2$, are also now separated into two regions by the new line. Since we introduce 2 new regions but lose 1 older one, we have:



$L_n = L_{n-1} + 2n - n = L_{n-1} + n$, for $n \geq 1$.

Then, $L_n = L_{n-1} + n = L_{n-1} + (n-1) + n = \ldots$

$= L_0 + 1 + 2 + \cdots (n-1) + n = 1 + \frac{1}{2}n(n+1)$.

• Print the power set of a given set $\{a_1, ..., a_n\}$. Given the power set of $\{a_1, \ldots, a_{n-1}\}$, P_{n-1}, the power set of P_n is $P_{n-1} \cup \{\{X, a_n\}, X \in P_{n-1}\}$. The base case is $P\emptyset = \{\emptyset\}$.

```
PowerSet(S)
if size(S) == 0
   return {0}
else
   P0 = PowerSet(S(1:n-1))
   for i=1:size(P0)
      P1(i) = [P0(i) S(n)]
   end
   return P0 Union P1
```

• Tower of Hanoi: Given three pegs A,B,C, and a tower of n disks, initially stacked in decreasing size on peg A. Objective: transfer the entire tower from A to C. Rules: (a) move only one disk at a time, (b) never put a larger disk on top of a smaller. [Historical note: $n = 8$, original puzzle by Lucas. $n = 64$ the Tower of Brahma]

Example with 3 disks: (1-2-3). Move 1 to C. Move 2 to B. Move 1 to B. Move 3 to C. Move 1 to A. Move 2 to C. Move 1 to C. Thinking recursively: Move(1:n,A,C):

```
Move(diskRange,fromPeg,toPeg,auxPeg)
    if size(diskRange)==1 disk 1 in fromPeg goes to toPeg
    else Move(1:n-1,fromPeg,auxPeg)
         Move(n:n,fromPeg,toPeg)
         Move(1:n-1,auxPeg,toPeg)
```

Let H_n be the total number of disk moves. Then $H_1 = 1$, $H_n = 2H_{n-1} + 1$, $n > 1$.
We can use the iterative method to solve the recurrence relation.

$$\begin{aligned} H_n &= 2H_{n-1} + 1 = 2(2H_{n-2} + 1) + 1 \\ &= 2^2 H_{n-2} + 2 + 1 = 2^2(2H_{n-3} + 1) + 2 + 1 \\ &= 2^3 H_{n-3} + 2^2 + 2 + 1 \\ &= \cdots \\ &= 2^{n-1} H_1 + 2^{n-2} + \cdots + 2^2 + 2 + 1 \\ &= 2^{n-1} + \cdots + 2^2 + 2 + 1 \\ &= 2^n - 1 \end{aligned}$$

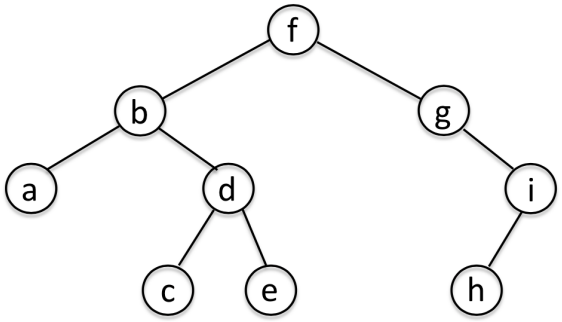
Fun video on Towers of Hanoi: https://www.youtube.com/watch?v=2SUvWfNJSsM&t=502s

• Binary Tree Traversals

```
Print_Inorder(root)
    if (root == None) return
    Print_Inorder(root.left)
    print(root)
    Print_Inorder(root.right)

Print_Prefix(root)
    if (root == None) return
    print(root)
    Print_Prefix(root.left)
    Print_Prefix(root.right)

Print_Postfix(root)
    if (root == None) return
    Print_Postfix(root.left)
    Print_Postfix(root.right)
    print(root)
```




```
Inorder: a b c d e f g h i

Prefix:  f b a d c e g i h

Postfix: a c e d b h i g f
```

• The Josephus problem: During the Jewish-Roman war, Josephus was among a band of 41 Jewish rebels trapped in a cave by the Romans. Preferring suicide to capture, the rebels decided to form a circle and, proceeding around it, to kill every third remaining person until no one was left. But Josephus, along with a friend, wanted none of this suicide nonsense; so he quickly calculated where he and his friend should stand in the vicious circle. The end of the story is that both Josephus and his friend lived and Josephus later became a famous historian.

In our variation, we start with n people numbered 1 to n around a circle, and we eliminate every second remaining person until only one survives. Problem is to determine the survivor's inital number in the list, $J(n)$.
Example with $n = 10$. Then, the elimination order is 2, 4, 6, 8, 10, 3, 7, 1, 9, or

```
1  2  3  4  5  6  7  8  9  10
   X     X     X     X     X          so 5 survives, i.e., J(10) = 5.
      X           X
X                       X
```

We can define J recursively. Clearly $J(1) = 1$.
Consider a list with an even number, $n = 2k$. We first eliminate 2,4,...,2k, and the remaining indices are [1, 3, 5, ... , 2k-1] = 2[1, 2, 3, ... , k]-1. Thus any index x in [1...k] corresponds to the index 2*x-1 in the original list. So if we recursively solve the $J(k)$ problem (i.e., the Josephus index for the list [1...k]), we can obtain the index in the 2k list, i.e., $J(2k) = 2J(k) - 1$.
Consider a list with an odd number $n = 2k + 1$. We eliminate 2,4,6,...,2k, and the 1, so the remaining indices are [3, 5, ... , 2k-1, 2k+1] = 2[1, 2, 3, ... , k]+1. Thus any index x in [1...k] corresponds to the index 2*x+1. Again, if we recursively solve the $J(k)$ problem, we can obtain the index in the 2k+1 list: $J(2k+1) = 2J(k) + 1$.

$$J(1) = 1, \text{ and } J(n) = \begin{cases} 2J(n/2) - 1 & \text{if } n = 2k \\ 2J(\lfloor n/2 \rfloor) + 1 & \text{if } n = 2k+1. \end{cases}.$$

To solve this recurrence we look at the binary representation of $n = (b_m, \ldots, b_0)_2$, assuming $b_m = 1$. Then, $\lfloor n/2 \rfloor = (b_m, \ldots, b_1)_2$.
Moreover, whether the recursion uses the +1 or -1 depends on whether n is odd ($b_0 = 1$) or even ($b_0 = 0$). Since +1 or -1 can be written as $2b_0 - 1$ in each case, we have:

$$\begin{aligned}
J(n) &= J((b_m, \ldots, b_0)_2) = \\
&= 2J((b_m, \ldots, b_1)_2) + 2b_0 - 1 = 2\left(2J((b_m, \ldots, b_2)_2) + (2b_1 - 1)\right) + (2b_0 - 1) \\
&= 2^2 J((b_m, \ldots, b_2)_2) + 2^2 b_1 + 2b_0 - 2 - 1 \\
&= 2^3 J((b_m, \ldots, b_3)_2) + 2^3 b_2 + 2^2 b_1 + 2b_0 - 2^2 - 2 - 1 \\
&\cdots \\
&= 2^m J((b_m)_2) + 2^m b_{m-1} + \ldots + 2^3 b_2 + 2^2 b_1 + 2b_0 - 2^{m-1} - 2^2 - 2 - 1 \\
&= 2^m + 2(2^{m-1} b_{m-1} + \ldots + 2^2 b_2 + 2^1 b_1 + b_0) - (2^m - 1) \\
&= (b_{m-1}, \ldots, b_0, 0)_2 + 1 \\
&= (b_{m-1}, \ldots, b_0, 1)_2
\end{aligned}$$

Based on this, we can see that for $0 \le l < 2^m, J(2^m + l) = 2l + 1$, as shown in the Table:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	$16 \cdots$
$J(n)$	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	$1 \cdots$

• Rabbits and the Fibonacci numbers: A young pair of rabbits (one of each sex) is placed on an island. A pair of rabbits does not breed until they are 2 months old. After they are 2 months old, they produce a new pair each month. What is the total number of rabbits on the island after n months, assuming no rabbits ever die?

Month	Mature pairs	Young pairs	Total pairs
0	0	0	0
1	0	1	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8

Let f_i be the number of pairs after i months. Then $f_0, f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

Finding the closed form of the Fibonacci numbers:

Define function $F(x) = f_0 + f_1 x + f_2 x^2 + \cdots = \sum_{i \geq 0} f_i x^i$. Then

$$\begin{aligned} F(x) &= f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots \\ xF(x) &= f_0 x + f_1 x^2 + f_2 x^3 + \cdots \\ x^2 F(x) &= f_0 x^2 + f_1 x^3 + \cdots \end{aligned}$$

So $F(x) - xF(x) - x^2 F(x) = x$. Therefore, $F(x) = \frac{x}{1-x-x^2}$. Next we try to express $\frac{x}{1-x-x^2}$ as a power series, then the coefficient of the term x^n is the nth Fibonacci number.

First, observe that the two roots for $x^2 + x - 1 = 0$ are $\frac{-1 \pm \sqrt{5}}{2}$. So

$$\begin{aligned} 1 - x - x^2 &= -(x^2 + x - 1) \\ &= -(x - \frac{-1-\sqrt{5}}{2})(x - \frac{-1+\sqrt{5}}{2}) \\ &= -(x + \frac{1+\sqrt{5}}{2})(x + \frac{1-\sqrt{5}}{2}) \\ &= -(x+\phi)(x+\hat{\phi}), \end{aligned}$$

where $\phi = \frac{1+\sqrt{5}}{2}$ and $\hat{\phi} = \frac{1-\sqrt{5}}{2} = 1-\phi$. Now consider $\frac{x}{1-x-x^2}$.

$$\begin{aligned}
\frac{x}{1-x-x^2} &= -\frac{x}{(x+\phi)(x+\hat{\phi})} \\
&= \frac{x}{(1-\hat{\phi}x)(1-\phi x)} \\
&= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi x} - \frac{1}{1-\hat{\phi}x}\right) \\
&= \frac{1}{\sqrt{5}}\left(\sum_{n\geq 0}\phi^n x^n - \sum_{n\geq 0}\hat{\phi}^n x^n\right) \text{ (when } |x|<1) \\
&= \frac{1}{\sqrt{5}}\sum_{n\geq 0}(\phi^n - \hat{\phi}^n)x^n.
\end{aligned}$$

Therefore, the coefficient of x^n is

$$f_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right).$$

Counting, Lecture 15

Combinatorics: the study of arrangements of objects. Counting of objects with certain properties, is an important part of combinatorics, frequently used to determine the complexity of algorithms.

Basics of counting *Reading: Rosen 5.1*

• **Product rule:** Suppose that a procedure can be broken down into two tasks.
If (there are n_1 ways to do the first task) *and for each of these* ways
(there are n_2 ways to do the second task),
then there are $n_1 n_2$ ways to do the procedure.

Example: The number of license plates of three letters (a to z) followed by three numbers (0 to 9): $26^3 \cdot 10^3$.

Example: The number of functions from $A = \{a_1, \ldots, a_m\}$ to $B = \{b_1, \ldots, b_n\}$: n^m, since a_1 can be mapped in n ways, a_2 can be mapped in n ways independently, etc.

Example: The number of one-to-one functions from A to B: $n(n-1)\cdots(n-m+1)$.
n choices for a_1
$n-1$ choices for a_2
... $\quad\Longrightarrow\quad$ $n(n-1)...(n-m+1)$
$n-m+1$ choices for a_m

Example: The number of subsets of set S:
Since there is a one-to-one correspondence between a subset of S and a bit string of length $|S|$ (i.e., 1 then the element i belongs in the subset, 0 if it doesn't belong), then number of subsets of S is equal to the number of bit strings of length $|S|$, which is $2^{|S|}(= 2*2*....*2)$. This is also all the numbers from 0 to $2^{|S|-1}$.

• **Sum rule:** If a task can be done *either* in one of n_1 ways *or* in one of n_2 ways, where none of the set of n_1 ways is the same as any of the set of n_2 ways, $\Longrightarrow$ there are $n_1 + n_2$ ways to do the task.

Example: Choose one element from one of three lists of sizes n_1, n_2, n_3: $n_1 + n_2 + n_3$
Here is an example of counting using both rules in combination.

Example 5: The number of passwords that must be 6–8 characters long, where each character is an *uppercase letter or a digit*, and must contain *at least one digit*:
Let P_6, P_7, and P_8 be the numbers of passwords of length 6, 7, and 8, respectively. Since there is clearly no overlap between them, the total number of passwords is $P_6 + P_7 + P_8 = (36^6 - 26^6) + (36^7 - 26^7) + (36^8 - 26^8) = 2,684,483,063,360$.
Reason: All passwords with at least 1 digit = all passwords with letters or digits − all passwords without any digit (i.e., letters only)

• **The inclusion-exclusion principle:** If a task can be done in n_1 or n_2 ways, but there are m ways of the set of the n_1 ways which are also in the set of the n_2 ways, then the total number of ways to do the task is $n_1 + n_2 - m$.
Example 6: The number of 8-bit strings that start with 1 or end with 00:
$1 * * * * * * * : 2^7$ such strings
$* * * * * * 00 : 2^6$ such strings
$1 * * * * * 00 : 2^5$ such strings
Total: $2^7 + 2^6 - 2^5 = 128 + 64 - 32 = 160$.

Note the correspondance to set operations:
$|A \times B| = |A| * |B|$ ... product rule
$|A \cup B| = |A| + |B|$ ... sum rule
$|A \cup B| = |A| + |B| - |A \cap B|$ ... inclusion-exclusion principle

Reading: Rosen 5.2:
• **The pigeonhole principle:** If k is a positive integer and $k+1$ or more objects are placed into k boxes, then there is at least one box containing two or more objects.
Proof with contraposition: Assume the conclusion is not true:
$\neg(\exists$ box with number of objects $\geq 2) \equiv \forall$ box has objects ≤ 1. There are k boxes, so the total number of objects is $\leq k$, which is not the assumed $k+1$ objects.
Example: In a group of 367 people, there are at least two people with the same birthday.

•**The generalized pigeonhole principle:** If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ objects.
Proof by contradiction. Denote by b_i is the number of objects in box i. Asssume not the conclusion: for all boxes, $b_i \leq \lceil \frac{N}{k} \rceil - 1$. Then $\sum_{i=1}^{k} (\lceil \frac{N}{k} \rceil - 1) = k\lceil \frac{N}{k} \rceil - k$.
However, $\lceil \frac{N}{k} \rceil < \frac{N}{k} + 1$ (e.g., 2 = $\lceil 4/2 \rceil < 4/2 + 1 = 3$ or 3 = $\lceil 5/2 \rceil < 5/2 + 1 = 3.5$)
Then, $k\lceil \frac{N}{k} \rceil - k < k(\frac{N}{k} + 1) - k = N$, a contradiction because there are N objects.
Example: Among 100 people, there are at least $\lceil 100/12 \rceil = 9$ who were born in the same month.
Example: For any positive integer n, there is a multiple of n that contains only 0s and 1s.
Proof: For positive integer n, consider the following $n+1$ integers,

$$1, 11, 111, \ldots, 11 \cdots 1,$$

where the last number contains $n+1$ 1s. Note that there are n remainders, $0, \ldots, n-1$, when an integer is divided by n (ie., mod $n \in \{0, \ldots, n-1\}$). By the pigeonhole principle, among the $n+1$ numbers in our list, there must be two numbers, say a and b with $a > b$, with the same remainder when divided by n. So $a - b$ is divisible by n, thus a multiple of n and $a - b$ contains only 0s and 1s.

Example: In a month of 30 days, a baseball team plays at least one game a day but no more than 45 games in the month. Show that there must be a period of some number of consecutive days during which the team plays exactly 14 games.

How is this possible? E.g., (16,16,0,...) not possible. I have 30 1's: 1 1 1 ... 1. I want to show that it's not possible to add up to 15 more ones while avoiding having 14 consecutive ones.

Proof: Let g_i = number of games on i-the day. Because $g_i \geq 1$ (one game per day), there are only 15 games remaining, so $1 \leq g_i \leq 16$.

Let a_j be the number of games played *on or before* the jth day of the month.

Then $a_1 < a_2 < \ldots < a_{30}$ is an increasing sequence of distinct positive numbers, with $1 \leq a_j \leq 45$.

Moreover, $a_1 + 14 < a_2 + 14 < \ldots < a_{30} + 14$ is also an increasing sequence of distinct positive numbers, with $15 \leq a_j + 14 \leq 59$.

The 60 positive integers $(a_1, a_2, \ldots, a_{30}, a_1 + 14, a_2 + 14, \ldots, a_{30} + 14)$ are all less than or equal to 59.

By the pigeonhole principle, $2 = \lceil 60/59 \rceil$ of these integers are equal. Since a_i and $a_i + 14$ are distinct, one of the two numbers must be a_i and the other must be $a_k + 14$, with $a_i = a_k + 14$. This means that exactly 14 games are played from day $k+1$ to day i.

Example: Show that among any $n+1$ positive integers not exceeding $2n$ there must be an integer that divides one of the other integers.

Proof: Assume the $n+1$ numbers are $a_1, a_2, \ldots, a_{n+1}$. Each can be expressed a power of 2 times an odd number (including 1). Thus $a_j = 2^{k_j} q_j$ for $j = 1, 2, \ldots, n+1$, where k_j is a nonnegative integer and q_j is an odd positive number less than $2n$. Since there are only n odd positive numbers less than $2n$, then by the pigeonhole principle, there must be two of $q_1, q_2, \ldots, q_{n+1}$, say q_i and q_m, that are equal. Let $q = q_i = q_m$. Then $a_i = 2^{k_i} q$ and $a_m = 2^{k_m} q$. Then it is either a_i divides a_m or a_m divides a_i.

Example 6: Assume in a group of six people, any pair of individuals are either friends or enemies. Show that there are three mutual friends or three mutual enemies in the group.

Proof: Let A be one of the six. The remaining can be either friends of enemies to A. According to the generalized pigeonhole principle, the number of each group must be $\geq \lceil 5/2 \rceil = 3$.

Assume there are three or more friends of A. Call B,C,D three of the friends of A.

− If any two of B, C, D are friends, then these two and A form a group of three mutual friends.

− If no one is a friend of the other among B, C, D, then (B, C, D) is a group of three mutual enemies.

If we assume there are three or more enemies of A, the proof is similar.

The **Ramsey number** $R(m,n)$, where $m,n \geq 2$ are positive integers, denotes the minimum number of people at a party such that there are either m mutual friends or n mutual enemies, assuming any pair of people at the party are either friends or enemies. The above example shows that $R(3,3) \leq 6$. In fact, it can be proved that $R(3,3) = 6$.

Permutations and combinations, Lecture 16

Reading: Rosen 5.3 and 5.5

• **Permutation** is the *ordered* arrangement of distinct elements.

An r-permutation is an ordered arrangement of r elements of a set. The number of r-permutations of a set of n elements is denoted by $P(n,r)$.

For $1 \leq r \leq n$, we have to fit elements in r locations without repetition. n choices for the first location, $n-1$ for the second, and so on, thus:

$$P(n,r) = n(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}.$$

Note: $P(n,n) = n! = n!/0! = n!/1$ is the usual permutations of n elements

Example: How many ways can we select 3 students from a group of 5 to award gold, silver, and bronze medals? $P(5,3) = 5 \cdot 4 \cdot 3 = 60$.

Example: How many permutations of the letters $ABCDEFGH$ contain the string ABC? We can treat ABC as one letter, then we get permutations of 6 symbols: $6! = 720$.

• **Combination:** An r-combination of elements of a set is an *unordered* selection of r elements from the set, i.e., simply a subset of size r.

The # of r-combinations of a set with n distinct elements is denoted by $C(n,r)$, also $\binom{n}{r}$.

Theorem. For $0 \leq r \leq n$, we have

$$C(n,r) = \frac{n!}{r!(n-r)!} = C(n,n-r)$$

Proof: Note that all r-combinations can be obtained by taking first all r-combinations of a set, and then for each combination take all permutations of its elements, then

$$P(n,r) = C(n,r) \cdot P(r,r) \quad \text{or} \quad C(n,r) = \frac{P(n,r)}{P(r,r)} = \frac{n!}{r!(n-r)!}.$$

Finally note that when we select r of the n elements, we implicitly select the rest $n-r$ to exclude from our subset, so $C(n,r) = C(n,n-r)$.

Example: How many committees of three students can be formed from a group of five students (no order between committee members)? $C(5,3) = 10$.

Example: How many committees can be formed with 3 professors from the 9-faculty math department and 4 professors from the 11-faculty CS department?

$C(9,3) \cdot C(11,4) = 84 \cdot 330 = 27720$.

Teaser: How many committees of 6 profs from math and 7 profs from CS?

• **Permutations with repetition.** Ordered arrangements where an element can appear multiple times.
Each of the r locations may have one of n elements, thus by the product rule, the # of r-permutations of a set of n elements with repetition is: n^r.
Example: How many string of length 10 can be formed from the lower-case English alphabet? 26^{10}.

• **Combinations with repetition:** The number of ways to get an *unordered* selection of r elements out of n types of elements with the elements of the same type repeating.
E.g., Select 4 coins out of nickels (N), dimes (D), and quarters (Q)

```
4N          4D          4Q
3N 1D       3D 1Q       3Q 1N
3N 1Q       3D 1N       3Q 1D
2N 2D       2D 2Q       2Q 2N
2N 1D 1Q    2D 1N 1Q    2Q 1N 1D
```

15 choices (since the order does not matter). A complicated approach would be: for each of i=0...4 cases of nickels, there are 0...4-i cases of dimes, leaving one 1 posibility for quarters. However, if the types of coins is large, the number of nested summations is large and difficult.
We model the problem differently. Represent each of the 4 coins as a star (*). We have 3 boxes (one for N,D,Q) that we want to put the 4 stars in, say $\boxed{**}\boxed{*}\boxed{*}$ for 2N,1D,1Q. Because order does not matter (all coins of the same type are the same), the 3 boxes can be represented by two separators $|$:
$**|*|*$ means 2N,1D,1Q
$**||**$ means 2N,2Q
$||****$ means 4Q
Theorem. The # of r-combinations from a set with n types of unlimited number of elements with repetition is

$$C(r+n-1,r) = C(r+n-1,n-1) = \frac{(r+n-1)!}{r!(n-1)!}$$

Proof 1: We have r stars for each element and $n-1$ separators for each of the n types of elements. Each sequence of r stars and $n-1$ separators defines a way of selection. This is the number of ways to place the r stars in $(r+n-1)$ positions, which is therefore $C(r+n-1,r)$ or $C(r+n-1,n-1)$.
Proof 2: We want all possible permutations of $r+n-1$ stars and separators: $(r+n-1)!$. However, since all stars are identical and so are the separators, we have overcounted the permutations by $r!$ and by $(n-1)!$. We must divide: $(r+n-1)!/(r!(n-1)!)$.

Example: How many different ways are there to buy six cookies from a store that sells four types of cookies? $C(6+4-1,6) = C(9,6) = C(9,3) = \frac{9\cdot 8\cdot 7}{3\cdot 2\cdot 1} = 84$.
Example: How many different dozens of doughnuts can be made when there are 8 varieties?
$C(12+8-1,12) = 19!/12!/7! = 50388$.
Example: How many nonnegative integer solutions exist for $x_1+x_2+x_3 = 11$?
Think of the problem as three bins (i-th bin for x_i), each containing different number of ones. Altogether we need $r = 11$ ones. We need to bin them in $n = 3$ bins: 11111|1111|11
So the total number is $C(11+2,2) = C(13,2) = \frac{13\cdot 12}{2\cdot 1} = 78$.
Example: What is value of k printed below?

```
k=0
for i(1) = 1 to p
   for i(2) = 1 to i(1)
      for i(3) = 1 to i(2)
         ...
         for i(m) = 1 to i(m-1)
            k=k+1
print(k)
```

Note that k=k+1 occurs for every combination of 1≤i(m)≤i(m-1)≤ ⋯ ≤ i(1)≤ p. How many such sequences can we have?
Choose m integers from $1,2,3,\ldots,p$, which can be the same (i.e., there is repetition), and in increasing order, e.g., (5,6,7,7,8,8,9,...). Then assign them to i(m) to i(1): e.g., i(m)=5, i(m-1)=6, ..., i(1) = largest. This assignment to indices is unique.
We look to find all the ways of picking such m integer sequences.
Take p-1 stars and separate them with m bars.
Set the j-th number = 1 + the number of stars on the left of the j-th bar.
E.g., $|**|**|*||**$ denotes the selection of numbers (1,3,5,6,6)
Thus, we have n=(m+1) bins (m separators) and r=p-1 stars:

$$k = C(r+n-1,r) = C(p-1+m,p-1) = C(p-1+m,m).$$

Permutations and combinations, continued, Lecture 17

• **Permutation with indistinguishable objects:** Want the number of permutations of n elements, where there are n_1 indistinguishable elements of type 1, n_2 indistinguishable elements of type 2, ..., and n_k indistinguishable elements of type k such that $n_1 + n_2 + \cdots + n_k = n$.
Ex: the number of permutations of BOB is not 3! Only BBO, BOB, OBB, because permutations of BB are indistinguishable.
Theorem: # perms with indistinguishable objects is $\frac{n!}{n_1!n_2!\cdots n_k!}$.
Proof 1 We have n posititions. First place n_1 elements of type 1 in $C(n,n_1)$ ways. This leaves $n-n_1$ positions free. Then the n_2 elements of type 2 can be placed in $C(n-n_1,n_2)$ ways. The n_3 type 3 elements can be placed in $C(n-n_1-n_2,n_3)$ ways, and so on until type k. Then by the product rule the number of requested permutations is

$$C(n,n_1)C(n-n_1,n_2)\cdots C(n-\sum_{i=1}^{k-1} n_i, n_k)$$

$$= \frac{n!}{n_1!\cancel{(n-n_1)!}}\frac{\cancel{(n-n_1)!}}{n_2!\cancel{(n-n_1-n_2)!}}\cdots\frac{\cancel{(n-\sum_{i=1}^{k-2} n_i)!}}{n_{k-1}!\cancel{(n-\sum_{i=1}^{k-1} n_i)!}}\frac{\cancel{(n-\sum_{i=1}^{k-1} n_i)!}}{n_k!0!} = \frac{n!}{n_1!n_2!\cdots n_k!}$$

Proof 2 There are $n!$ permutations of all elements. However, if we freeze all types except type i, there are $n_i!$ permutations of type i elements that are indistinguishable and counted for each permutation of the frozen types. Thus, we must divide $n!$ by $n_i!$ for all i.

Example: How many different strings can be made by reordering the letters of the word SUCCESS? $\frac{7!}{3!2!1!1!} = 420$.
Note that the second proof of number of r-combinations with repetition is based on the fact that there are $n-1$ indistinguishable bars and r indistinguishable stars.

• **Distributing objects into boxes:** Some counting problems can be modeled as enumerating the ways *objects* can be placed into *boxes*, with the following 4 possibilities:

objects, boxes $\in$ { distinguishable, indistinguishable }.

- **Distinguishable objects into distinguishable boxes:** We distribute distribute n distinguishable objects into k distinguishable boxes so that n_i objects are placed into box i, where $i = 1,2,\ldots,k$, with $\sum n_i = n$.

  Theorem # of ways to place distinguishable objects into distinguishable boxes = # perms of indistinguishable objects or $= \frac{n!}{n_1!n_2!\cdots n_k!}$.

Proof: First choose n_1 objects from n to be placed in box 1, then choose n_2 objects from $n-n_1$ to be placed in box 2, and so on. By the product rule, the numbers of ways is $C(n,n_1)C(n-n_1,n_2)\cdots C(n-n_1-\cdots-n_{k-1},n_k) = \frac{n!}{n_1!n_2!\cdots n_k!}$.

Example: How many ways are there to distribute hands of 5 cards to each of four players from a standard deck of 52 cards? $\frac{52!}{5!5!5!5!32!}$. Note that 32! is needed because we do not care how the remaining cards in the deck are ordered (and should not count all their permutations).

- **Indistinguishable objects into distinguishable boxes:** The number of ways to distribute n indistinguishable objects into k distinguishable boxes is the same as the number of ways of choosing n objects from a set of k types of objects with repetition allowed, which is equal to $C(k+n-1,n)$.

  Example: How many ways are there to place 10 indistinguishable balls into 8 distinguishable bins? $C(8+10-1,10) = C(17,10) = C(17,7) = \frac{17!}{10!7!} = 19448$.

  Caution: do not get confused with what goes in the "choose" part. Think of stars and dashes; the formula denominator is (stars! dashes!)

Indistinguishable boxes are hard problems.

- **Distinguishable objects into indistinguishable boxes:** Consider an example first.

  Example 12: How many ways are there to put 4 employees into three indistinguishable office? We name the employees by A,B,C,D. The question becomes: how many ways to partition set $\{A,B,C,D\}$ into three subsets?

  All four in one office: $0-0-4 \Rightarrow 1$ way

  Three in one office: $0-1-3 \Rightarrow C(4,3) = 4$ ways

  Two in one office and the other two in one office. Need all pairs from the 4 employees: (AB, CD), (AC, BD) (AD, CB), $0-2-2 \Rightarrow C(4,2)/2 = 3$
  (Why divide by 2? The choice of AB is the same as choice of CD).

  Two in one office and the other two each in one office: $1-1-2 \Rightarrow 6$
  This now is exactly $C(4,2)$, since there is no symmetry.

  So the final answer is $1+4+3+6 = 14$.

  There is no closed formula for the number of ways to distribute n distinguishable objects into k indistinguishable boxes. However, there is a complicated one.

  Let $S(n,j)$ be the number of ways to distribute n distinguishable objects into j indistinguishable boxes so that no box is empty. Therefore, the number we want is $\sum_{j=1}^{k} S(n,j)$.

In the above example, $S(4,1) = 1, S(4,2) = 4+3 = 7, S(4,3) = 6$. It can be shown that

$$S(n,j) = \frac{1}{j!}\sum_{i=0}^{j-1}(-1)^i\binom{j}{i}(j-i)^n.$$

So

$$\sum_{j=1}^{k} S(n,j) = \sum_{j=1}^{k}\frac{1}{j!}\sum_{i=0}^{j-1}(-1)^i\binom{j}{i}(j-i)^n.$$

- **Indistinguishable objects and indistinguishable boxes:** No closed form.

  Example 13: How many ways are there to pack six copies of the same book into four identical boxes, where a box can hold as many as six books?

  We enumerate ways of packing: (6), (5,1), (4,2), (3,3), (4,1,1), (3, 2,1), (2, 2, 2), (3, 1, 1, 1), (2, 2, 1, 1). So the answer is 9.

  Distributing n indistinguishable objects into k indistinguishable boxes is the same as writing n as a sum of at most k positive integers in non-increasing order, i.e., find k $a_i \geq 0 : n = \sum_{i=1}^{k}$.

  Note: This is not the same as the problem with the stars and bars $\sum_{x_i} = n$, because the x_i were distinguishable, i.e., $x_1 = 3, x_2 = 1$ is different from $x_2 = 3, x_1 = 1$. In our case, a_i are not.

  There is no simple closed formula for this number.

Summary

Without repetition	with repetition
r-perms $P(n,r) = \frac{n!}{(n-r)!}$	r-perm with rep n^r
r-combs $C(n,r) = \binom{n}{r} = \frac{n!}{r!(n-r)!}$	r-combs with rep (n types) $C(r+n-1,r)$

Objects into boxes

n dist $\rightarrow k$ dist $\equiv$ r-perms with k types of indist. obj: $\frac{n!}{n_1!\cdots n_k!}$
n indist $\rightarrow k$ dist $\equiv$ r-combs with repet. $r = n$ ind.obj. $types = k$: $C(n+k-1,n)$

n dist $\rightarrow k$ indist $\equiv \sum_{j=1}^{k} S(n,j)$ as above
n indist $\rightarrow k$ indist $\equiv$ no closed form

Binomial coefficients. Lecture 18

Reading: Rosen 5.4
The number of combinations $C(n,r) = \binom{n}{r}$, is also called a **binomial coefficient** since it occurs as coefficient in the expansion of powers of binomial expressions such as $(x+y)^n$.
• **The Binomial Theorem:** Let x and y be variables and let n be a nonnegative integer.

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i}y^i = \binom{n}{0}x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n-1}xy^{n-1} + \binom{n}{n}y^n.$$

Proof: Write $(x+y)^n$ as $(x+y)(x+y)\cdots(x+y)$. Each term in the product when it is expanded is of the form $x^{n-i}y^i$ for $i = 0,1,2,\ldots,n$. To count the number of terms of the form $x^{n-i}y^i$, note that to obtain such a term it is necessary to choose $n-i$ x's from the n factors of $(x+y)$ in the product so that there are i y's in the term. Therefore, the coefficient of $x^{n-i}y^i$ is $\binom{n}{n-i}$, which equals $\binom{n}{i}$.

Example: What is the expansion of $(x+y)^4$?
$(x+y)^4 = \binom{4}{0}x^4 + \binom{4}{1}x^3y + \binom{4}{2}x^2y^2 + \binom{4}{1}xy^3 + \binom{4}{4}y^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$.

Example: What is the coefficient of $x^{12}y^{13}$ in the expansion of $(2x-3y)^{25}$?
$\binom{25}{13}(2x)^{25-13}(-3y)^{13} = \binom{25}{13}2^{12}(-3)^{13}x^{12}y^{13}$. So the coefficient is $\binom{25}{13}2^{12}(-3)^{13}$.

• **Corollaries:**
(1) $\sum_{k=0}^{n}\binom{n}{k} = 2^n$ (e.g., $x = 1 = y$. Note that summing the numbers of all sets
(2) $\sum_{k=0}^{n}(-1)^k\binom{n}{k} = 0$.
(3) $\sum_{k=0}^{n}2^k\binom{n}{k} = 3^n$.

• Identities of binomial coefficients:
Pascal's identity: $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$ for positive integers $n \geq k$.

Vandermonde's identity: $\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{r-k}\binom{n}{k}$ for nonnegative integers m,n,r with r not exceeding either m or n.

Graphs. Terminology, definitions, motivation. Lecture 19

Reading: Rosen 9.1-9.4

Definition 1: A **graph** or an **undirected graph** $G = (V, E)$ consists of V, a set of **vertices** or **nodes**, and E, a set of **edges**, where an edge $e = (u, v)$ connects its **endpoins** u and v.

Definition 2: A **directed graph** $G = (V, A)$ consists of V, a set of vertices, and A, a set of **arcs**, where an arc $a = (u, v)$ goes from initial vertex u to terminal vertex v. Note that arc (u, v) and arc (v, u) are different.

Other types of graphs:

- Simple graph vs. multigraph
- Weighted graph vs. non-weighted graph
- Graph with self-loops

Examples of use of graphs:

- Distance maps, Road maps. Possible questions:

  Find a path between nodes (cities). Find minimum/maximum path. Find a circuit that visits all cities. Find a circuit of minimum weight that visits all cities.

- Social Networks: Friendship edges. Study groups with special interests. Uncover social trends. Discover threats, etc.

  Six degrees of separation (the average distance in the graph of connections between any two people in the world)

- Collaboration graphs

  the Hollywood graph with more than 800,000 vertices (actors) and more than 20 million edges (appearing in the same film).

  Erdos Number = minimum path of Author(i) to Paul Erdos.

- Call Graphs: A directed edge (u, v) if phone number u called number v. Additional information on edges (date, time, duration). E.g., common benchmark: ATT 20 day call graph has 290M vertices and 4B edges.

- The Web-Graph. A web-page is a vertex. Hyperlinks are directed edges between vertices (the links from any website to any other website). We can only estimate the current size of the Web-graph (close to 6B vertices and more than 120B edges).

  Google searches/ web crawlers

  If $A \to B$, then A is somehow relating to B. A search must show this (although we should be aware that correlation does not imply causation). On the other hand, if $A \to B \to C \to D \to E$, A is also related to E but remotely. A search should weigh this less. This gives rise to the Science of Searching.

- Data mining of graph data: Model relationships with edges and try to find unexpected connections. E.g., It was observed that those who buy diapers in the Super Market, also buy beer! Super Markets made money locating beer next to diapers!

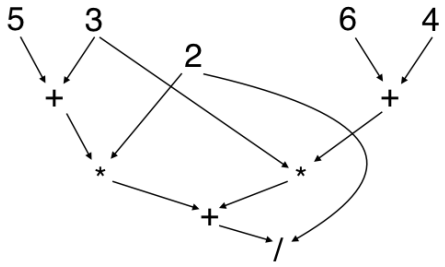
- Precedence constraints: Task = vertex. Edge = prerequisite.

  Our prerequisite graph for the CS major. vertex=course number, edge="is prerequisite for". Edges: (141,241), (141,243), (243,303), etc.

  Precedence graph in factories: (boil point, open air valve), (boil point, cool water), (air pressure > 100, run!)

- Trees are graphs without cycles.

  Compilers use parse trees to identify the order of evaluation of an expression.



  $((5+3)*2+3*(6+4))/2$

- Parallel processing: Schedule operations to be performed in parallel (see the parse tree above)

Definition: The **degree** of a node, $deg(v)$, in an undirected graph is the number of edges incident with it.

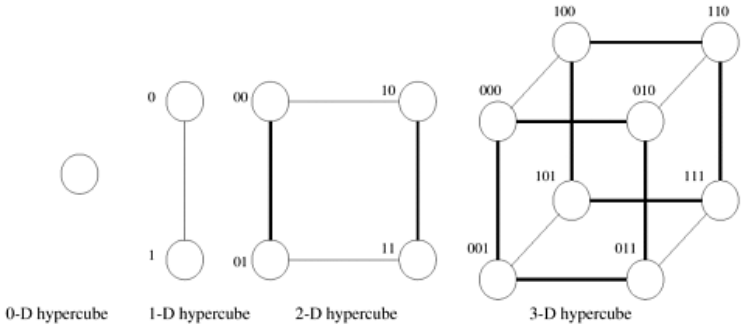
Handshaking Theorem: In an undirected graph $G = (V,E)$, $\sum_{v \in V} deg(v) = 2|E|$.
(proof: Every edge (u,v) is counted twice, once in the degree of u and once for v).

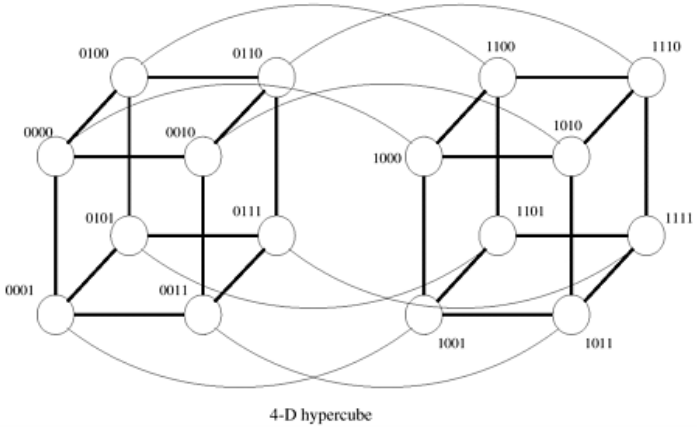
Theorem: An undirected graph has an even number of nodes with odd degree.
Proof. We know that $2|E| = \sum_{v \in V} deg(v) =$ even. Split the $V = V_{even} \cup V_{odd}$ of even and odd degree nodes. If the V_{odd} is odd, then $\sum deg(v) = \sum_{v \in V_{even}} deg(v) + \sum_{v \in V_{odd}} deg(v) = even + odd * odd = odd$. Contradiction.

In a directed graph, the **in-degree** of a node, $deg^-(v)$, is the number of arcs entering the node and the **out-degree** of a node, $deg^+(v)$, is the number of arcs leaving the node.
Theorem: In a directed graph $G = (V,A)$, $\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = |A|$.

Some special graphs:

- A **complete graph** is one in which there is an edge between every pair of vertices. For such a graph with n nodes, there are $\binom{n}{2} = \frac{1}{2}n(n-1)$ edges. Notice also that this is the arithmetic sequence summation (first node connects to n-1, second to n-2, and so on).
- A **cyclic graph:** $C_n, n \geq 3$. If v_i vertices, the edges are $(v_1,v_2),(v_2,v_3),\ldots,(v_n,v_1)$.
- A **hypercube** or **n-Cube**, Q_n, is an n-dimensional hypercube, a graph that has vertices representing the 2^n bit strings of length n.
  Property: Two vertices in Q_n are adjacent iff their bit strings differ in exactly one position.





- A **bipartite graph** $G = (V,E)$, where V is partitioned into V_1 and V_2, and for any $e = (v_1,v_2)$, there must be $v_1 \in V_1$ and $v_2 \in V_2$.

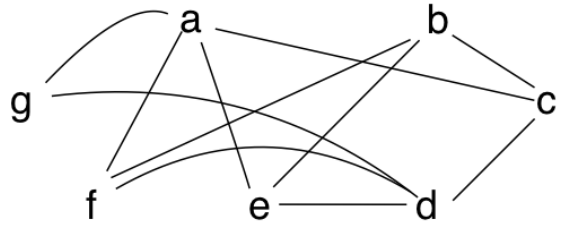
  Theorem: G is bipartite if and only if G is 2-colorable, i.e., nodes in G can be colored with two colors such that no two nodes which are connected by an edge

are given the same color.

Example: C_6 is bipartite. Color red has (0,2,4) and color black has (1,3,5).

Example: C_3 is not bipartite!

Example: 2-colorable / bipartite with (a,b,d) and (c,e,f,g)



Representing graphs:

- **Adjacency list:** In an undirected graph, for each vertex, create a list of all vertices connected to it by an edge. For example,

Vertex	Adjacent vertices
a	b,c,e
b	a
c	a,d,e
d	c,e
e	a,c,d

  In a directed graph, for each vertex, create a list of all vertices reachable from it by an arc. For example,

Initial vertex	Terminal vertex
a	b,c,d,e
b	b,d
c	a,c,e
d	
e	b,c,d

- **Adjacency matrix:** For a graph $G=(V,E)$ with $V=\{v_1,v_2,\ldots,v_n\}$, create an $n\times n$ matrix $A=[a_{ij}]$ such that $a_{ij}=1$ if $(v_i,v_j)\in E$ and $a_{ij}=0$ if $(v_i,v_j)\notin E$.

  For the undirected graph represented by the first adjacency list, using v_1,v_2,v_3,v_4,v_5 to replace the names of a,b,c,d,e respectively, its adjacency matrix is

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

For the directed graph represented by the second adjacency list, its adjacency matrix is

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

- **Incidence matrix:** For an undirected graph $G = (V, E)$ with $V = \{v_1, \_ 2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$, create an $n \times m$ matrix $A = [a_{ij}]$ such that $a_{ij} = 1$ if e_j is incident with v_i and $a_{ij} = 0$ if e_j is not incident with v_i.

  For the graph represented by the first adjacency list earlier, assuming v_1, v_2, v_3, v_4, v_5 are a, b, c, d, e respectively and $e_1 = (a, b), e_2 = (a, c), e_3 = (a, e), e_4 = (c, d), e_5 = (c, e), e_6 = (d, e)$, its incidence matrix is

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Graph connectivity. Lecture 20

Definition: A **path** is a sequence of nodes, where there is an edge between any two consecutive nodes in the sequence. A **cycle** is a path that begins and ends at the same node. A path or cycle is **simple** if it does not contain the same node more than once.
Definition: An undirected graph is **connected** if there is a (simple) path between every pair of distinct nodes. A **connected component** of a graph is a connected subgraph that is not a proper subgraph of another connected subgraph.
Definition: A directed graph is **strongly connected** if there is a path from a to b and from b to a for any nodes a and b in the graph. A directed graph is **weakly connected** if the undirected graph obtained after removing directions on all arcs is connected. A **strongly connected component** of a directed graph is a strongly connected subgraph that is not contained in a larger strongly connected subgraph.

0.1 Euler and Hamilton paths/cycle

Reading: Rosen 9.5
Euler paths and cycles:

- An **Euler path** is a path that contains each edge exactly once. An **Euler cycle** is a cycle that contains each edge exactly once.
- The seven bridges of Konigsberg, a problem solved by Swiss mathematician Leonhard Euler.
- **Theorem:** A connected multigraph with at least two nodes has an Euler cycle if and only if each of it nodes has an even degree.
- **Theorem:** A connected multigraph has an Euler path but not an Euler cycle if and only if it has exactly two nodes of odd degree.

Hamilton paths and cycles:

- A **Hamilton path** is a path that passes throught each vertex exactly once. A **Hamilton cycle** is a cycle that passes through each vertex exactly once.
- A game called the "Icosian Puzzle", invented by Irish mathematician Sir William Rowan Hamilton, where there is a dodecahedron (a polyhedron with 20 vertices and 12 regular pentagons as faces) with each vertex representing a city and the objective of the puzzle is to travel along the edges to visit each city once to come back to the first city to make a tour.

- **Dirac's Theorem:** If G is a simple graph with $n \geq 3$ vertices such that $deg(v) \geq \frac{n}{2}$ for any vertex v in G, then G has a Hamilton cycle.
- **Ore's Theorem:** If G is a simple graph with $n \geq 3$ vertices such that $deg(u) + deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton cycle.
- The best algorithms **known** for finding a Hamilton path/cycle in a graph or determining no such path/cycle exists have exponential-time complexity, as these problems have been proved to be **NP-complete**, a class of problems for which no polynomial-time algorithms have been found.
- The **Traveling Salesman Problem** is a generalization of the Hamilton cycle problem, where the input is a weighted graph and the goal is to find a hamilton cycle with the minimum total weight.

0.2 A collection of graph problems

- Graph traversal: depth-first search and breadth-first search
- Topological sort in a directed acyclic graph:
- Shortest paths in a weighted graph: one-to-one, one-to-all, and all-to-all, Dijkstra's algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm
- Minimum spanning tree: Kriskal's algorithm and Prim's algorithm
- Network flow problem: Ford and Fulkerson algorithm
- Bipartite matching:
- Vertex cover, clique, dominating set, independent set:
- Graph coloring: