Discrete Structures Reference Sheet

Ref: http://oeis.org/wiki/List_of_LaTeX_mathematical_symbols

Last Updated: February 6, 2017

$\forall$	for all
$\exists$	exists
$\therefore$	therefore
$\rightarrow$	implies
$\to$	also implies (prof uses this one)
$\leftrightarrow$	if and only if (abbreviated iff)
$\leftrightarrow$	also iff
$\oplus$	XOR, exclusive or
$\odot$	Boolean product
$\equiv$	is equivalent to
$\wedge$	and
$\vee$	or
$\neg$	not
$\lfloor x \rfloor$	floor of x. Next integer smaller than or equal to x
$\lceil x \rceil$	ceiling of x. Next integer larger than or equal to x

Sets:

- Two sets are **disjoint** if they share no common elements. (p.128, 130)
  Disjoint sets A and $B \to A \cap B = \varnothing$
- $a \in A$: a is an element of the set A
- $a \notin A$: a is not an element of the set A
- $A \subseteq B$: A is a subset of B
  $\varnothing \subseteq A$: the empty set is a subset of all sets
  $x \in A \to x \in B$
- $A \nsubseteq B$: A not a subset of B
  $x \in A \not\to x \in B$
- $A \subset B$: A is a proper subset of B (implies $A \neq B$)
  $x \in A \to x \in B$
- $A \not\subset B$: A not a proper subset of B (implies $A \neq B$).
  $x \in A \not\to x \in B$
- $A \cup B$: the union of A and B. Contains all elements in either A or B
  $= \{x | x \in A \vee x \in B\}$
  $x \in A \to x \in (A \cup B)$
  $x \in (A \cup B) \not\to x \in A$
- $A \cap B$: the intersection of A and B. Contains all elements in both A and B
  $= \{x | x \in A \wedge x \in B\}$
  $x \in A \not\to x \in (A \cap B)$
  $x \in (A \cap B) \to x \in A$
- $\bar{A}$: the compliment of A. $\bar{A}$ contains every element that is not in A
  $x \in \bar{A} \leftrightarrow x \notin A$
  $\bar{A} = \{x | x \notin A\}$
  $\bar{A} = U - A$
- $A = B$: Sets A and B are equal (contain the same elements)
  Does not matter if elements are in a different order or repeated
  $\to \forall x (x \in A \leftrightarrow x \in B)$
  $\to (A \subseteq B) \wedge (B \subseteq A)$
- $A \times B$: The Cartesian product of sets A and B. Contains tuples of all combinations of the elements in A and B
  $A \times B = \{(a, b) | a \in A \wedge b \in B\}$
- $C = A - B$: C contains all elements of A that are not in B
  $C = \{x | x \in A \wedge x \notin B\}$
- **Special sets:**subsubsectionSpecial sets
  - $\mathbb{U}$: Universal set. Contains all elements (no really, ALL of them)
  - $\mathbb{N}$: set of all natural numbers (positive integers, and 0 depending on who you ask)
  - $\mathbb{Z}$: set of all integers
  - $\mathbb{Q}$: set of all rational numbers
  - $\mathbb{R}$: set of all real numbers
  - $\varnothing$: the empty set
    $\varnothing \subseteq A$: the empty set is a subset of any sets
  - $\mathcal{P}(A)$: Power set of A. Contains all possible subsets of A, including A, $\varnothing$.
    $|\mathcal{P}(A)| = 2^{|A|}$
    $X \subseteq A \leftrightarrow X \in \mathcal{P}(A)$

Functions:

- let f be a function that maps elements ftom A to B, where A and B are sets. (p.141)
  - We write this function as $f : A \to B$. if $f(a) = b$, then b is the image in B of a.
- **domain:** the set of values for which f is defined. (A in above)
- **co-domain:** the set of values for which f maps elements to. (B in above)(not the same as the range)
- **one-to-one:** the function maps every element in A to a unique element in B. (AKA injective)
- **onto:** the function contains a mapping for every element in B. (AKA surjective)
  - Even if a function is onto, then mappings need not be unique, e.g. $(f(a) = x \wedge f(b) = x) \not\to a = b$
- **bijection:** one-to-one and onto
- **Function Composition:**
  - let $f : A \to B$ and $g : B \to C$

- $\rightarrow (f \circ g) : A \rightarrow C$
- $\rightarrow (f \circ g)(a) = f(g(a))$ for all $a \in A$

- **Inverse Functions:**subsubsectionInverse Functions
  - An inverse function only exists if a function is a bijection (both one-to-one and onto). For the previous example function, the inverse of f is $h : B \rightarrow A$

Cardinality:

- The number of elements in a set. Can be finite, countably infinite, or uncountably infinite
  - $|A|$ cardinality (number of elements) of A
- $|A| = |B|$ if and only if there exists a bijection (one-to-one and onto) function mapping A and B
- $|A| \leq |B|$ if and only if there exists a one-to-one function mapping A and B
- A set is countable if:
  - it has a finite number of elements
  - it can be listed without skipping elements
  - it can be demonstrated to have the same cardinality as $\mathbb{N}$ (set of positive integers)
- $\aleph_0$ cardinality of an infinite set (aleph sub 0)
- $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$ are countably infinite
- $\mathbb{R}$ is uncountably infinite

Sequences:

- A sequence is an infinite ordered list of elements
- $a = \{a_1, a_2, a_3, ...a_n, ...\}$
- Geometric progression: $\{a, ar, ar^2, ...ar^n, ...\}$
- Arithmetic progression: $\{a, a + d, a + 2d, ...a + nd, ...\}$
- Recursive sequence: a sequence where each element is defined in terms of other elements
  - example: **Fibonacci Sequence:** $f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$
  - solving a recursive sequence means finding valid initial conditions
- **Summations:**subsubsectionSummations
  - $\cup_{i=1}^{n} A_n$ union of all sequences A_n. This is the set of all elements of every set
  - $\cap_{i=1}^{n} A_n$ intersection of all sequences A_n. This is the set of all elements common to all sets A_n
  - $\sum_{i=0}^{n} a_n$ sum of the sequence of elements a_n
  - $\sum_{j \in S} a_j$ sum of the sequence of elements a_j, whose indexes are elements of the set S
  - $\prod_{i=0}^{n} a_n$ just like normal summation notation, except this is the product of all terms
  - $\vee_{i=0}^{n} q_n$ logical OR of all logical propositions a_n. True if **any** a_n is true.
  - $\wedge_{i=0}^{n} q_n$ logical AND of all logical propositions a_n. True if **all** a_n are true.
- **Strings:**subsubsectionStrings
  - A string is a finite sequence of characters from a finite alphabet
  - The empty string is λ (lambda)
  - The set of all possible strings of any length is countably infinite

Algorithms:

- An algorithm is a finite list of precise instructions to solve a problem in a domain
  - can be defined in English or pseudocode
  - if a function is not computable, there cannot exist an algorithm that computes that function
- **Computability:**subsubsectionComputability
  - A function is computable if there exists a computer program that can find values of the function. A function is not computable if such a computer program does not exist.
- We are often interested in the number of operations required to complete a given task using a given algorithm
- Common algorithm tasks include searching and sorting
- **Greedy Algorithm:** an algorithm that decides the best choice at each individual decision.
  - does not necessarily choose the best (most optimal) overall choice
- **Halting Problem:** Does there exist an algorithm that takes as input an algorithm and the parameters for that algorithm, and determines whether the given algorithm will eventually stop with this input.
  - i.e. an algorithm that tests for infinite loops
  - in 1936, Alan Turing proved that this problem is not computable (p201)

Complexity:

- **Time Complexity:** the number of operations needed to complete the operation
  - Usually given as worst-case complexity (another useful case is the average case complexity)
- **Big-O notation:**subsubsectionBig-O notation
  - $f(x)$ is $O(g(x))$ if $|f(x)| \leq C|g(x)|$ for $x > k$
    - i.e. f grows slower than a fixed multiple of g as x approaches ∞
  - C and k are called witnesses. We only need one pair of them for which the relationship is true to show that the relationship is always true
- **Big-Ω notation:**subsubsectionBig-Ω notation
  - (big-omega notation)
  - $f(x)$ is $\Omega(g(x))$ if $|f(x)| \geq C|g(x)|$ for $x > k$
    - i.e. f grows faster than a fixed multiple of g as x approaches ∞
  - $f(x)$ is $\Omega(g(x))$ if and only if $g(x)$ is $O(f(x))$
- **Big-Θ notation:**subsubsectionBig-Θ notation
  - (big-theta notation)
  - $f(x)$ is $\Theta(g(x))$ if and only if $f(x)$ is $O(g(x))$ and $\Omega(g(x))$
    - If true, $f(x)$ and $g(x)$ are said to be the same order

Theorems:

- If A and B are countable sets, $A \cup B$ is also countable (p174)
- If A and B are sets with $|A| \leq |B|$ and $|B| \leq |A|$, then $|B| = |A|$ (p174)
  - i.e. there exists a one-to-one function mapping A to B, and a one-to-one function mapping B to A
  - (Schrder-Bernstein Theorem)
- Let $f(x) = a_n x^n + ... + a_1 x + a_0$ where a_n are real number coefficients. Then $f(x)$ is $O(x^n)$ (p209)
  - A polynomial of order n is $O(x^n)$
- The sum of first n positive integers is $O(n^2)$
- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$ (p213)
- If $f_1(x)$ and $f_2(x)$ are both $O(g(x))$, then $(f_1 + f_2)(x)$ is $O(g(x))$ (p213)
- $f_1(x)f_2(x)$ is $O(g_1(x)g_2(x))$ if $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$
- If $d > c > 1$: n^c is $O(n^d)$ but n^d is not $O(n^c)$ (p212)
- If $b > 1$ and c and d are > 0, $(log_b n)^c$ is $O(n^d)$, but n^d is not $O((log_b n)^c)$ (p212)
- If $d > 0$ and $b > 1$, n^d is $O(b^n)$ but b^n is not $O(n^d)$ (p212)
- If $c > b > 1$, b^n is $O(c^n)$, but c^n is not $O(b^n)$ (p212)
- $n!$ is $O(n^n)$
- $\log(n!)$ is $O(n \log(n))$. (logarithmic identity)
- Any constant c is $O(1)$

Propositional Logic:

- a logical proposition is a statement that can be either **true** or **false**
- Traditional propositional variables are $p, q, r, s \ldots$
- **Operators:**subsubsectionOperators

Negation	$\neg p$	not p
Conjunction	$p \wedge q$	p and q
Disjunction	$p \vee q$	p or q
Implication	$p \rightarrow q$, $p \rightarrow q$	if p then q
Biconditional	$p \leftrightarrow q$, $p \leftrightarrow q$	p if and only if q, p iff q

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

- Operator Precedence:
  - parenthesis ()
  - $\neg$
  - $\wedge, \vee$
  - $\rightarrow, \leftrightarrow$
- **Converse:** the same statement, but backwards.
  if $p \rightarrow q$ then $q \rightarrow p$
- **Inverse:** is negation of both sides of the statement.
  if $p \rightarrow q$, then $\neg p \rightarrow \neg q$
- **Contrapositive:** is equivalent to the inverse of the converse. (or equivalently, the converse of the inverse)
  if $p \rightarrow q$, then $\neg q \rightarrow \neg p$
- **Tautology:** A statement is a tautology if it can be shown to be true by definition
- **Contradiction:** always false, never true
- **Contingency:** neither a tautology nor a contradiction
- Two compound statements are logically equivalent ($\equiv$) if they have the same truth table.
- **Propositional Satisfiability:** A statement is propositionally satisfiable if there exists a combination of inputs for which it is true. It is not propositionally satisfiable if and only if it's negation is a tautology
- **DeMorgan's Laws:**subsubsectionDeMorgan's Laws
  - **1.** $\neg(p \wedge q) \equiv \neg p \vee \neg q$
  - **2.** $\neg(p \vee q) \equiv \neg p \wedge \neg q$
- **Rules of Inference:** (p.72)

Counting:

- **Pigeonhole Principle:**subsubsectionPigeonhole Principle if N pigeons are placed into $N-1$ boxes, at least one box must have 2 or more pigeons
- **Generalized Pigeonhole Principle:**subsubsectionGeneralized Pigeonhole Principle if N objects are placed into k boxes, then there must exist a box with $\lceil N/k \rceil$ objects
  - Works regardless of the values of N and k
- **Permutations:**subsubsectionPermutations unique subsets of n things taken r at a time, where the order **does** matters
  - Assumes that duplicates cannot be selected
  - $P(n,r) = \frac{n!}{(n-r)!} = n(n-1)(n-2)(n-3)\ldots r$
  - **Permutations with duplicates:** $= n^r$
- **Combinations:**subsubsectionCombinations unique subsets of n things taken r at a time, where the order **does not** matter
  - $C(n,r) = \frac{n!}{r!(n-r)!} = \frac{P(n,r)}{P(r,r)} = \frac{n(n-1)(n-2)(n-3)\ldots r}{r!}$
  - **Combinations with duplicates:** $= \frac{(n+r-1)!}{r!(n-r)!}$
- **Product Rule:**subsubsectionProduct Rule if there is a task of two subtasks tasks which can be done n_1 and n_2 ways respectively, then the overall task can be done $n_1 n_2$ ways
- **Sum Rule:**subsubsectionSum Rule if a task can be done either n_1 or n_2 ways, and there is no overlap between the ways, the task can be done $n_1 + n_2$ ways

- **Inclusion-Exclusion principle:**subsubsectionInclusion-Exclusion principle $|A \cup B| = |A| + |B| - |A \cap B|$

Recurrence Relations:

- **Linear Homogeneous:**subsubsectionLinear Homogeneous in the form $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$ where $c_1, c_2, \ldots c_k$ are real numbers and $c_k \neq 0$

- **Degree** is k. The relation is defined in terms of the previous k terms of the sequence. (and therefore the terms before a_{n-k} don't matter to a_k)
- linear because all previous terms of a have exponent 1 and constant coefficient
- homogeneous because all terms are previous terms of a. (no constant numbers, no direct expressions of n)
- **Theorem 1 Solution:**
  (when there are multiple roots)
  Characteristic equation: $r^k - c_1 r^{k-1} - c_2 r^{k-2} - \ldots - c_{k-1} r - c_k = 0$
  Find roots $r_1, r_2, \ldots r_k$ of characteristic equation, and then use $a_n = \alpha_1 r_1^n + \ldots + \alpha_k r_k^n$ with base the conditions to find $\alpha_1 \ldots \alpha_k$.
  Final solution equation is $a_n = \alpha_1 n^{r_1} + \ldots + \alpha_k n^{r_k}$
- **Theorem 2 Solution:**
  (when there is a single root)
  same as theorem 1, except find α using
  $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$

- **Linear Non-Homogeneous:**subsubsectionLinear Non-Homogeneous
- FIXME
- **Divide and Conquer Recurrence Relations:**subsubsectionDivide and Conquer Recurrence Relations
  (Theorem 1)
  - f is an increasing function $f(n) = af(n/b) + c$
    if $a > 1$: $f(n)$ is $O(n^{\log_b a})$
    if $a = 1$: $f(n)$ is $O(\log n)$
  - When $n = b^k$ and $a \neq 1$:
    $f(n) = C_1 n^{\log_b a} + C_2$
    $C_1 = f(1) + c/(a-1)$
    $C_2 = -c/(a-1)$
- **Master Theorem:**subsubsectionMaster Theorem
  (Theorem 2)
  - f is an increasing function $f(n) = af(n/b) + cn^d$
    if $a < b^d$: $f(n)$ is $O(n^d)$
    if $a = b^d$: $f(n)$ is $O(n^d \log n)$
    if $a > b^d$: $f(n)$ is $O(n^{\log_b a})$

Relations:

- a and b are related by R if $(a, b) \in R$, also written aRb
- a relation can be treated as a set of tuple and/or as a function
- a **binary relation** is a relation from one set to another
  a binary relation from A to B is a subset of $A \times B$
  ($R \subseteq A \times B$)
- a **relation on a set** is a relation from A to A

A relation R on a set A is: (p.576)

- **Reflexive** if:
  - $\forall(a \in A) : (a, a) \in R$
  - All nodes have a loopback
- **Symmetric** if:
  - $\forall(a, b \in A) : ((a, b) \in R) \leftrightarrow ((b, a) \in R)$
  - All streets are two-way
- **Antisymmetric** if:
  - $\forall(a, b \in A) : ((a, b) \in R \wedge (b, a) \in R) \rightarrow a = b$
  - One-way streets only
- **Transitive** if:
  - $\forall(a, b, c \in A) : ((a, b) \in R \wedge (b, c) \in R) \rightarrow (a, c) \in R$
- **Irreflexive** if:
  - $\forall(a \in A) : (a, a) \notin R$
- **Asymmetric** if:
  - $((a, b) \in R \rightarrow (b, a) \notin R)$
  - Asymmetric $\leftrightarrow$ Antisymmetric and Irreflexive
- relations can be combined in any way that sets can be combined (union, intersection, subtraction...)
- **composite relation** of R and S
  sets A, B, C
  relation $R : A \rightarrow B$, $S : B \rightarrow C$
  $S \circ R = \{(a, c) | a \in A \wedge c \in C \wedge \exists b((a, b) \in R \wedge (b, c) \in S)\}$
  Note that S and R are reversed
  - powers of a relation R on a set: $R^1 = R$, $R^2 = R \circ R$, $R^3 = R \circ R \circ R = R^2 \circ R \ldots$
  - a relation R on a set is transitive if and only if $R^n \subseteq R$ for $n \geq 1$

Matrices:

- Matrix relation: $m_{ij} \begin{cases} 1 \text{ if } (a_i, b_j) \in R \\ 0 \text{ if } (a_i, b_j) \notin R \end{cases}$
- **Reflexive** diagonal ($i = j$) is all 1's $\begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$
- **Symmetric, Antisymmetric**
  Symmetric: all opposite diagonals are equal.
  Antisymmetric: opposite diagonals aren't both one
  $S : \begin{bmatrix} & 0 & & \\ 0 & & 1 & \\ & 1 & & 1 \\ & & 1 & \end{bmatrix} A : \begin{bmatrix} & 0 & & \\ 0 & & 0 & \\ & 1 & & 1 \\ & & 0 & \end{bmatrix}$
- **Boolean Product** $\odot$
  used to find the composite of two relations
  $M_{S \circ R} = M_R \odot M_S$
  Power of relation: $M_{R^n} = M_R^{[n]}$ boolean_product.png has gone missing, maybe eventually I'll fix this

Closures of Relations

- Closure: $R \cup$ minimum elements to add to R such that it satisfies a given

condition.
- E.g. The reflexive closure of R is the loopback arrows that are missing, plus R itself.

n-ary Relations:
- (for databases)
  width of table = degree of relation = n
  each record is an n-tuple
  elements of the n-tuple are fields of the database
- **Primary key:** element of the tuples such that each tuple contains a unique element in that position
  a field is a primary key if it is unique for all possible entries in the database
- **Selection operator:** returns all n-tuples in R that satisfy a condition C
- **Join operator:** joins two tables. $J_p(R, S)$ joins tables R and S, where there are p duplicate keys.
  Width of result=(width of R)+(width of S)$-p$
- **Projection:** $P_{i_1,i_2,\dots i_m}(R)$ returns a new table with only columns $i_1, i_2, \dots i_m$ from R. Note: $i_1, i_2, \dots i_m$ may be any subset of integers on range $[1, m]$

Equivalence Relation:
- A relation R is an equivalence relation if it is **reflexive**, **symmetric**, and **transitive** (p.608)
- if aRb, a are said to be equivalent, also written $a \sim b$

Partial Ordering
- A relation R on a set S is a partial ordering if the relation is **reflexive**, **antisymmetric**, and **transitive**.
- represents $\leq$, not $<$, since it is reflexive
- Poset: relation on it's set, denoted as a tuple: (R, S)

Hasse Diagram
- A visual representation of a partial ordering
- leaves out the edges in the relation that are required by the reflexive and transitive properties, leaving a clean graph that easily represents which nodes are related to which

Graphs:

Types of graphs:	Edges	multi?	loops
Simple Graph	undirected	no	no
Multgraph	undirected	yes	no
Pseudograph	undirected	yes	yes
Simple Directed graph	directed	no	no
Directed Multigraph	directed	yes	yes
Mixed Graph	both	yes	yes

Degree of vertex: $deg(v)$
- number of edges connecting to it (one loopback counts twice).
- an undirected graph has an even number of vertexes of odd degree
- **in-degree**: number of directed edges pointing into vertex. $deg^-(v)$
- **out-degree**: number of directed edges pointing out of a vertex. $deg^+(v)$
  for directed graph $G = (V, E)$: $\sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v) = |E|$

Special Graphs:
- K_n: every vertex is directly connected to every other vertex
- $C_n, (n \geq 3)$: vertexes connected in a loop
- $W_n, (n \geq 4)$: same as C_{n-1}, but with an extra vertex in the middle that is connected to all outer vertexes
- Q_n, n-dimensional Hypercube: represents all (2^n) bit strings of length n. vertexes are adjacent if their bit strings differ by only one bit (in any position)

Hamiltonian Path
- Path across a graph that visits every node exactly once
- Every node on a Hamiltonian path has exactly 2 edges on the path

Bipartite Graphs:
- graph $G = (V, E)$ is bipartite if you can split the vertexes into two subsets V_1, V_2 such that edges connect two points in one of the subsets. A graph is bipartite if you can assign one of two colors to every vertex and have no like color vertexes adjacent.

Complete bipartite graph:
- a bipartite graph where every node in V_1 is adjacent to every node in V_2

Adjacency Matrix:
- for a graph $G = (V, E)$ with $n = |V|$ vertexes, adjacency matrix m is $n \times n$ where $m_{i,j} = (1 \text{ if } (V_i, V_j) \in E, 0 \text{ otherwise})$

Incidence Matrix:
- for a graph $G = (V, E)$ with $n = |V|$ vertexes and $m = |E|$ edges, incidence matrix m is $n \times m$ where $m_{i,j} =$ (1 if edge e_j is incident with vertex v_i, 0 otherwise)

Graph Isomorphism:
- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a one-to-one function $f : V_1 \to V_2$ such that $\forall((a, b) \in E_1)((f(a), f(b)) \in E_2)$. The function f is called an isomorphism

Trees:
- height h = maximum path length starting at root node and ending at a leaf node
- internal vertex: has parent and at least one child
- leaf vertex: has no children
- m-ary tree: each node has at most m children
- full m-ary tree: every vertex either has m children or is a leaf
- ordered root tree: tree where the children of each node are listed in order

Properties:
- a tree with n vertexes has $n - 1$ edges
- a full m-ary tree with:
  - i internal vertexes contains $n = mi + 1$ vertexes
  - n vertexes has $i = (n-1)/m$ internal vertexes and $l = ((m-1)n + 1)/m$ leaves

    - l leaves has $n = (ml-1)/(m-1)$ vertexes and $i = (l-1)/(m-1)$ internal vertexes
- A rooted tree is balanced if all nodes are at levels h or $h-1$
- there are at most m^h leaves in a m-ary tree of height h
- for rooted m-ary tree:
    - $h \geq \lceil \log_m l \rceil$
    - if the tree is full and balanced: $h = \lceil \log_m l \rceil$

Spanning tree:

- For a simple graph G, the spanning tree of G is a tree containing every node in G
- A simple graph is connected if and only if it has a spanning tree

Depth-first search

- All edges in the resulting tree are tree edges
- All edges not in the resulting tree are back edges
    - all back edges connect ancestors to descendants or vice versa
    - (back edges never connect nodes that are the same distance from the root node)
- Algorithm:
  add current node w to T
  recursively visit every node adjacent to w that isn't already in the tree

Breadth-first search

- add w to T
  recurse
  for every vertex added at the previous level, add all nodes adjacent to them to T

Traversing Trees:

- preorder traversal:
    - visits root node, then recursively preorder traverses subtrees from left to right
- inorder traversal:
    - recursively visit first child nodes' subtree inorder, then visit root, then recursively visit remaining child nodes' subtrees
- postorder traversal:
    - recursively postorder traverses subtrees from left to right, then finally visit root node

Finite State Machine:

deterministic FSM is same stuff as learned in ECEN248
non-deterministic FSM:

- can't predict where it might go
- because there are duplicate conditions for edges
- useful for stuff that should be random (e.g. video game AI's)
- A NDFSM can also recognize a (regular) language
- a NDFSM can be transformed into a FSM using an algorithm (don't know how)

A FSM can be used to recognize a pattern

- if the FSM terminates in a recognizing state (one with a double circle), then it recognizes the input
- if not, the input is not recognized
- useful for recognizing a grammar

Kleene Closure: of A is A^*

- all possible concatenations of elements of A
- $A^* = \bigcup_{i=0}^{\infty} A^i$
- Recursive definition:
  $V_0 = \{\lambda\}$
  $V_1 = A$ (the original set) (concatenate every element one with λ)
  $V_{i+1} = \{wv | w \in V_i \wedge v \in V\}$ (concatenate every element with every other element, and itself)
  (and let i approach ∞)

Phrase Structure Grammar:

- $G = (V, T, S, P)$:
    - V = vocabulary: finite, nonempty set of symbols
    - T = terminal symbols
    - S = start symbols
    - P = productions
- Vocabulary: set of all symbols in the language
- Derivation tree = parse tree
- word or sentence over vocabulary V: string of finite length containing elements of V
- the empty string (λ) contains no symbols
- set of all words over V is V^*
- a language is a subset of V^*
- $N = V - T$ = nonterminal symbols
- production example: $A \rightarrow Bc$
    - A can be replaced with B concatenated with c
- Language generated by G:
    - $L(G)$ = all strings or terminals that can be derived from the starting symbols in S (using the productions P)
    - $L(G) = \{w \in T^* | S \overset{*}{\rightarrow} w\}$
    - $S \overset{*}{\rightarrow} w$ means you can derive w from S in a finite number of steps

Backus-Naur Form (BNF):

- a form for writing productions in a grammar
- concatenate all statements starting with the same nonterminal, and separate the right hand sides with |
- all nonterminals are enclosed in angle brackets $\langle\rangle$
- ::= is used instead of $\rightarrow$
- Example:

Productions: $A \to a, A \to Aa, a \to Bc$
BNF: $\langle A \rangle ::= a |\, \langle A \rangle\, a |\, \langle B \rangle\, c$

Regular Expressions:

- A set representing a regular expression is a regular set
  - contains all the words in the language that match it
- $\varnothing$ and λ are regular expressions
- for a set I, x is a regular expression over I if $x \in I$
  - (this expression matches only the element x)
- for regular expressions $A, B : (AB), (A \cup B), A^*$ are regular expressions
- (AB) : concatenation of sets represented by A and B
  - matches only A immediately followed by B
- $(A \cup B)$: matching either expression A or expression B

Kleene's Theorem:

- A set is regular if and only if it can be recognized by a FSA
- A set is generated by a regular grammar if and only if it is a regular set

Other machine types:

- **Pushdown Automaton:** includes a stack
  you can push something onto the stack, and you can pop something from the stack
  stack is First In, Last Out (FILO)
  stack is assumed to have infinite capacity
  can recognize $\{0^n 1^n\}$ but not $\{0^n 1^n 2^n\}$ because it only has one stack
- **Linear Bounded Automaton:**
  can recognize $\{0^n 1^n 2^n\}$
- **Turing Machine:**
  most powerful machine we know
  can recognize all languages generated by phrase structure grammars

etc:

- Non-deterministic Turing Machine (NDTM) is a thing
  basically just a Turing machine with probability built into the decisions it makes
- P and NP Problems:
  P: solvable in polynomial time
  NP: non-deterministically solvable in polynomial time
  All NP problems are checkable in polynomial time