Overview of Chapter/Sections 4.1, 6.1-3, and 7.2-4

April 9, 2021

1 Section 4.1

- The notation of sets (definition, order, cardinality, empty set, power set, Cartesian products, countable, uncountable, ...)
- Using predicate logic to determine when two sets are equal
- Relationships between sets
- binary and unary operations (and conditions for their proper definition)
- intersection, union, complements, set-difference, and Venn diagrams
- That set theory and propositional logic share a parallel set of laws (commutativity, associativity, distributivity, identities, complements); furthermore, there are two "dual" sets of laws. In some sense, union and intersection parallel logical or/and; set complement parallels logical negation; and the universal and empty sets parallel true and false.
- one-to-one correspondence, and proving that sets are the same size.
- The power set of a set is always bigger than the set itself. The Power set of a set with n elements has cardinality 2^n. The empty set, with 0 elements, has a power set of 1 element – the empty set itself. This holds true even for infinite sets, which tells us that infinity comes in an infinite number of larger and larger sizes.
- The natural numbers is an infinite set, the smallest (called $\aleph_0$ – "aleph null"). Yet it is the same size as all of the integers, the even natural numbers, the prime numbers, the rational numbers, and all "denumerable" infinite sets – those that can be put in one-to-one correspondence with the natural numbers.
- Therefore, we discover that, for infinite sets, **a proper subset may be the same size as the set itself**. However, a proper subset can never be **bigger** than the set itself.

2 Section 6.1

- Graph definitions and terminology: loops, parallel edges, directed, simple, complete, cycle, connected, etc.
- Special graphs (K_n, $K_{m,n}$)
- Isomorphic graphs:
  - **Definition**: Two graphs (N_1, A_1, g_1) and (N_2, A_2, g_2) are **isomorphic** if there are bijections (one-to-one and onto mappings) $f_1 : N_1 \to N_2$ and $f_2 : A_1 \to A_2$ such that for each arc $a \in A_1$, $g_1(a) = \{x, y\} \iff g_2[f_2(a)] = \{f_1(x), f_1(y)\}$ (replace braces by parentheses for a directed graph).
  - **Theorem**: Two simple graphs (N_1, A_1, g_1) and (N_2, A_2, g_2) are isomorphic if there is a bijection $f : N_1 \to N_2$ such that for any nodes n_i and n_j of N_1, n_i and n_j are adjacent $\iff f(n_i)$ and $f(n_j)$ are adjacent.
  - Tests for when two graphs are **not** isomorphic.
- Planar graphs (one which can be drawn in two-dimensions so that its arcs intersect only in nodes)
- Euler's Formula for connected planar graphs states that

  $$r - a + n = 2$$

  where n is the number of nodes, a is the number of arcs, and r is the number of regions (including the infinite region surrounding the graph).
- Any graph failing to be planar has a subgraph isomorphic to either K_5 or $K_{3,3}$ (Kuratowski's Theorem).
- Computer representations of graphs:
  - the adjacency matrix, and
  - the adjacency list.

  (and advantages of one representation over the other).

3 Section 6.2

- **tree**: an acyclic, connected graph with one node designated as the **root** node (or defined recursively).
- tree terminology: root, binary, parent, child, leaf, etc.
- examples of trees
- tree representations

- tree traversal algorithms:

$$\begin{array}{r|ccc} preorder & root & left & right \\ inorder & left & root & right \\ postorder & left & right & root \end{array}$$

- expression trees: infix, prefix, postfix

4 Section 6.3

- **decision tree**: a tree in which
  - internal nodes represent actions,
  - arcs represent outcomes of an action, and
  - leaves represent final outcomes.
- Examples
- Lower Bounds on Searching
  - a. Any binary tree of depth d has at most $2^{d+1} - 1$ nodes. (Proof: look at the full binary tree, as it has the most nodes per depth.)

    b. Any binary tree with m nodes has depth $d \geq \lfloor \log m \rfloor$.
  - **Theorem** (on the lower bound for searching):

    Any algorithm that solves the search problem for an n-element list by comparing the target element x to the list items must do at least $\lfloor \log n \rfloor + 1$ comparisons in the worst case.
- Binary Search Tree (Binary Tree Search - follows the same path as an algorithm as the tree creation process!)
- Sorting
  - Theorem on the lower bound for sorting: you have to go to at least a depth of $\lceil \log n! \rceil$ in the worst case.

5 Section 7.2

- **Euler Path**: a path in which each arc is used exactly once ("highway inspector problem").
- **"Hand-shaking" Theorem**: in any graph, the number of odd nodes (nodes of odd degree) is even.
- **Theorem**: an Euler path exists in a connected graph $\iff$ there are either two or zero odd nodes.
- Using the EulerPath algorithm (simply counts up elements in a row i of the matrix (the degree of node i), and checks whether that's even or odd; if in the end there are not zero or two odd nodes, there's no Euler path!)
- **Hamiltonian Circuit**: a cycle using every node of the graph ("travelling salesman problem").

6 Section 7.3

- Shortest Path algorithms (for a simple, positively weighted, connected graph)
  - Dijkstra's Algorithm
  - Bellman-Ford Algorithm
  - Floyd's algorithm
- Minimal Spanning Trees: A **spanning tree** for a connected graph G is a non-rooted tree containing the nodes of the graph and a subset of the arcs of G. A **minimal** spanning tree is a spanning tree of least weight of a simple, weighted, connected graph G.
  - Prim's algorithm
  - Kruskal's algorithm

7 Section 7.4

Traversing a graph (generalizes tree traversal):

- depth-first strategy
- breadth-first strategy

Graph-traversal algorithms can be used as tree-traversal algorithms, too!

Remember: We use the convention that, given a choice, we should choose nodes in alphabetic order.