Overview of Chapters 4.1, 6.1-6.3, , and 7.2-7.3

April 7, 2025

Abstract

Your test will resemble the first exam. You are allowed a sheet of notes, front and back. It will be stapled to the exam before you begin.

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
- one-to-one correspondence, and proving that cardinalities are the same
- the cardinality of the power set is always greater than the power of the set itself: $Card(\wp(S)) = 2^{Card(S)}$

  From which we can conclude that there are infinitely many larger and larger infinities!

2 Section 6.1

- Graph definitions and terminology (e.g. simple, complete, degree of a vertex, 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).
- Computer representations of graphs:
  - the adjacency matrix, and
  - the adjacency list.

  (advantages of one 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
- examples of trees
- tree representations
- tree traversal algorithms:

$preorder$	$root$	$left$	$right$
$inorder$	$left$	$root$	$right$
$postorder$	$left$	$right$	$root$

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, which is order $n \log(n)$.

5 Section 7.2

- **Euler Path**: a path in which each arc is used exactly once.
- **Handshaking 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.
- The EulerPath algorithm shows that the problem of determining whether an Euler path exists is easy: 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.

  This is a hard problem, which we solved using symmetry and exhaustion.

6 Section 7.3

- Shortest Path algorithms (for a simple, positively weighted, connected graph)
  - Dijkstra's Algorithm
  - Bellman-Ford Algorithm
  - Floyd's algorithm