Skillset for Final

CS/ECE 374A: Intro. Algorithms & Models of Computation (Fall 2022) Ver: 1.01

The final exam is cumulative and will test material covered in the entire course. Since this is a multiple choice exam, some skills are less relevant.

Post midterm 2 skillset:

1. Turing Machines and Complexity Classes
   (a) Definition of a TM at a high-level and equivalence with programs. We will not ask you to design TMs for any concrete problem.
   (b) Definition of decidable languages.
2. NP, NP-Completeness and Polynomial-time Reductions
   (a) Understanding and designing polynomial-time certifiers/verifiers.
   (b) Definitions of NP, NP-Complete, NP-Hard
   (c) Knowledge of standard NP-Complete problems: SAT, 3SAT, CircuitSAT, Independent Set, Clique, Vertex Cover, Hamiltonian Cycle/Path in directed/undirected graphs, 3Color, Color.
   (d) Ability to prove that a given problem is in NP
   (e) Ability to prove that a given problem is NP-Hard via a polynomial time reduction from an existing NP-Hard problem from the given list.
   (f) Understand the definition of a polynomial-time reduction and its implications.
   (g) Ability to prove correctness of reductions
   (h) Understand basic boolean logic and properties of SAT/CircuitSAT formulas to enable reductions
   (i) Understanding Self reductions – solving NP-Hard problems using oracles for the decision problem. Being able to construct self reductions.
3. Algorithms and properties of minimum spanning trees
   (a) Understanding safe edges and unsafe edges, and how to use them in computing MST.
   (b) Cut and cycle properties to understand when a spanning tree is an MST.
   (c) Standard algorithms for MST: Kruskal, Prim, Borůvka. Run times and high-level implementation ideas.

Midterm 2 skillset:

1. Divide and Conquer Paradigm
   (a) Solving recurrences characterizing the running time of divide and conquer algorithms.
   (b) Familiarity with specific Divide and Conquer Algorithms and the running times: Binary Search, Merge Sort, Quick Sort, Karatsuba's Algorithm, Linear Selection.
   (c) Ability to design and analyze divide and conquer algorithms for new problems.
2. Backtracking and Dynamic Programming Algorithms
   (a) Using the dynamic programming methodology to design algorithms for new problems.
   (b) Ability to analyze the running time of dynamic programming algorithms.
3. Graphs
   (a) Basic definitions of undirected and directed graphs, DAGs, paths, cycles.
   (b) Definitions of reachable nodes, connected components, and strongly connected components.
   (c) Understand the structure of directed graphs in terms of the meta-graph of strongly connected components.
   (d) Understand the structure of DAGs: sources, sinks and topological sort.

4. Graph Search
   (a) Understand properties of the basic search algorithm and its running time.
   (b) Understand properties of depth first search traversal on directed and undirected graph.
   (c) Understand properties of the depth first search tree.
   (d) Understand properties of depth first search traversal on directed and undirected graph.
   (e) Algorithms based on search for finding connected components in undirected graphs, checking whether a graph is a DAG, topological sort for DAGs, knowledge of a linear-time algorithm to create the meta-graph, finding a cycle in a graph etc.
5. Shortest Paths in Graphs
   (a) Understand properties of the breadth first search tree.
   (b) Understand properties of breadth first search traversal on directed and undirected graph to find distances in unweighted graphs.
   (c) Dijkstra's algorithm for finding single-source shortest paths in undirected and directed graphs with non-negative edge lengths.
   (d) Negative length edges and Bellman-Ford algorithm to check for negative length cycles or find shortest paths if there is none.
   (e) Single-source shortest paths in DAGs — linear time algorithm for arbitrary edge lengths.
   (f) Shortest path trees and their basic properties.
   (g) Dynamic programming for shortest path problems in graphs.
6. Graph reductions and tricks
   (a) Modeling problems via graphs and solving them using graph structure, reachability and shortest path algorithms.
   (b) Adding sources, sinks, splitting edges, nodes
   (c) Creating layered graphs

Midterm 1 skillset:

1. Basic mathematics
   (a) Comfort with set notation, especially set operations like cross product and power set. Should know how to read and understand formally described sets, and should be able to describe new sets precisely.
   (b) Familiarity with alphabets, strings, and languages.
   (c) Ability to critically evaluate proofs and write proofs, especially induction proofs.
   (d) Ability to comprehend inductive definitions.
2. Formal models of computation (regular expressions, DFAs, NFAs, CFGs)
   (a) Understand formal definitions of machines, grammars and expressions. Be able to execute machines on simple examples, and infer if strings belong to sets defined by expressions/grammars. Understand what it means for a language to be described/accepted by a computational model.
   (b) Ability to design machines/grammar/expressions to describe/accept languages. Ability to formally describe them.
3. Transformations between computational models
   (a) Familiarity with proofs transforming NFAs to DFAs, and regular expressions to NFAs. Ability to carry out these constructions on examples.
   (b) Familiarity with the cross product construction to run multiple machines simultaneously.
   (c) Know asymptotic bounds of the resulting automata constructed by these transformations.

(d) Ability to perform new transformations on automata to prove regularity or construct automata/expressions with special properties.

4. Closure properties

   (a) Know standard closure properties (concatenation, union, intersection, complementation, set difference, Kleene star, reverse) for regular languages covered in lectures, labs and homework. Understand the proofs for these properties.

   (b) Know how to prove new closure properties either through automata transformations or using previously established closure properties.

5. Non-regularity

   (a) Ability to distinguish regular and non-regular languages

   (b) Ability to prove languages to be non regular using the fooling set argument. Know how to prove lower bounds on the number of DFA states using the fooling set argument as well.