Discrete Structures 2824 – A brief walk through the semester

Note that this is an incomplete guide through the material we have covered in CSCI 2824. It is not meant to be an exhaustive listing of material to study for the final exam. Also note that the "Example problems" are not meant to indicate the level of difficulty or particularly important applications or problem types. They are just meant to jog your memory.

- **Binary Arithmetic**
    - Converting integers and fractions from decimal to binary
    - Originally, we started using these tools because operating bit-wise makes sense from a computational perspective. But it turns out that binary is also a useful introduction to propositional logic, where propositions take one of two values, True or False
    - How many bits does it take to encode a piece of information?
    - **Example problem**: How long of a bit string do we need in order to give each of 30 people an ID number?
- **Propositional Logic and Applications**
    - This was our first step into formally "thinking like a computer" (at least for this class). Computers don't see any shade of gray – that bit is either a 0 or a 1 – it's either True or False. And so, using propositional logic, we were able to construct propositional statements and assess their truth values. We used these to solve puzzles and represent declarative statements, which we later used to construct arguments and proofs.
    - Logical connectives and compound propositions
    - Truth values and truth tables
    - Conditionals, and conditionals in English language (necessary and/or sufficient conditions, "only if")
    - Logic Puzzles – Knights and Knaves! Who wants coffee?
    - **Example problem**: Knights and Knaves. Satisfiability.
- **Logical Equivalences** – from propositional logic, we found that – just like in the spoken word – two statements might appear/sound different, but they mean the same thing
    - Logical equivalences were our first step into the world of manipulating propositional statements, which ultimately led to us writing out full-blown proofs.
    - Propositional Satisfiability – satisfiability connects our propositional statements to reality – is it actually possible for this complicated compound proposition to be true?
    - **Example problem**: Logical equivalences proofs, like using truth tables to show that $p \implies q \equiv \neg p \vee q$ (…or much more complicated things, using chains of logical equivalences)
- **Predicate Logic**
    - Turns out, propositional logic lacked the flexibility to make many statements that we would find useful in computational sciences. For example, it would be cumbersome to say "all computer science instructors ride a unicorn to work every morning and enjoy asparagus" via propositional logic. But predicate logic – by combining propositional functions and quantifiers, can handle these things!
    - Nested Quantifiers
        - Get comfortable translating to/from quantifiers from/to English
    - **Example problem**: You and your friend are at the dog park, and he tells you this about dogs: "For all of the dogs in this park, it is the case that dogs with brown fur always have a long tail." Let the domain of discourse be all dogs in the park and translate your friend's rule into a quantified statement. Be sure to define all propositional functions you might use.

- **Rules of Inference**
    - Convenient "mini-proofs" that we found to be useful again and again in the next section (Proofs), especially by linking together common sets of propositions and compound propositions.
    - **Example problem**: Pretty much any of our proofs – these and our logical equivalences are the steps that we keep writing out as justification in the right-hand column of the proof "stencil".
- **Proofs**
    - Proof Methods and Strategies (direct/conditional proof, proof by contradiction, proof by cases, contrapositive proof, proof by construction)
    - We brought everything from propositional logic and predicate logic together here.
    - **Example problem**: Prove that n is odd if and only if $n^2 - 4n + 3$ is even.
- **Set Theory and Set Operations**
    - Sets are among the most useful discrete structures. The propositional/predicate logic that we learned previously all used sets. For example, if we said that the domain of discourse was all CU students, we were actually saying that our propositional function has the domain that is the set of all CU students.
    - This was also a natural place to go after quantifiers and proofs because we sometimes would like to prove something is true (or false) about all or some element(s) in a set (universal and existential quantifiers, respectively)
    - Set unions, intersections, complements, differences
    - Empty sets, subsets, elements of sets vs subsets
    - Set identities
    - **Example problem**: Let A = {1, 2, 3, 4, 5} and B = {0, 3, 6}. Find B-A.
- **Functions and Cardinality**
    - One-to-one and onto functions
        - **Example problem**: Consider the function $f(n) = 2n^3$, where f: Z to Z. Which of the following choices best describes the function f(n)? (i) one-to-one, (ii) onto, (iii) neither, (iv) both
    - Inverse Functions and the composition of functions
    - We studied sets and then functions because functions operate on elements of one set (the domain) and map them into another set (the codomain).
    - Functions also turned out to be a useful tool to measure how large sets are (i.e. their cardinality)
    - Other important functions (floor, ceiling, etc.) – because these appear all over the place, and turn out to be very useful, especially when defining a one-to-one function from the set of natural numbers to a set of interest.
    - Countable and Uncountable Sets
    - **Example problem**: Let B be the set of all bit-strings that do not contain any 0's. What is the cardinality of B?
- **Sequences**
    - A natural next stop from functions because determining that a set is countable means we can define a sequence of its elements. Sequences are like functions from the natural numbers to some set.
    - **Example problem:** Find a closed form solution to the recurrence relation: $a_n = 2a_{n-1} + 3$, $a_0 = 1$

- **Algorithms**
  - Searching Algorithms – Linear Search, Binary Search
  - Sorting Algorithms – Bubble Sort, Insertion Sort
  - Greedy Algorithms
  - Get used to interpreting code, and thinking algorithmically
  - This followed nicely from sequences and functions because an algorithm uses the elements along a sequence as input to a function:

    Example:
    ```
    for ii in range(0,3):
            output[ii] = func(ii)
    ```
    This little algorithm does the function "func(...)" to elements from the set {0, 1, 2}.
- **Complexity**
  - Growth of Functions – big-), big-Omega, big-Theta
  - Worst-Case, Best-Case, Average-Case
  - **Example problem**: What is the smallest integer p such that $f(n) = 5n^3 - \log(n^2) + n^2 \log(n^4)$ is $O(n^p)$?
- **Matrix and Matrix Operations**
  - Matrix Addition and Multiplication (including matrix-vector multiplication)
  - Complexity of Matrix Addition and Multiplication
  - We care a great deal about these operations because they're so common in scientific computing, and potentially computationally expensive.
  - **Example problem**:

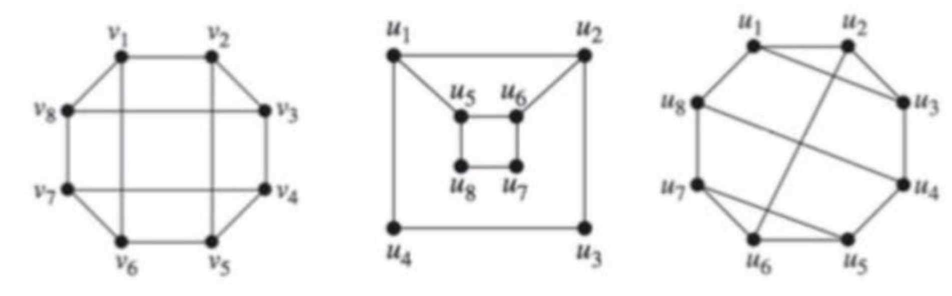
    Compute $A\mathbf{x}$, where

    $$A = \begin{bmatrix} 1 & 3 \\ -2 & 4 \\ 1 & 5 \end{bmatrix}, \text{and } \mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$
- **Induction**
  - We wanted to be able to prove things that should be true for an infinite number of cases, but we don't want to have to actually sit here and prove each and every case.
  - Induction allows us to show that a statement is true in general by showing:
    - (1) the statement is true for a Base Case (or a set of base cases), and
    - (2) if the statement is true for Case k, then it must be true for Case k+1
    - The induction hypothesis is "assume the statement is true for Case k" (weak induction", or "assume the statement is true for every case up through Case k" (strong induction)
  - Two forms: strong and weak – know which is which
  - **Example problem**: Use induction to prove that $1 + 2 + 3 + .. + n = n(n+1)/2$. Did you use strong or weak induction?
- **Combinatorics/Counting**
  - Sum rule / product rule
  - Pigeonhole principle
    - **Example problem**: Suppose you own a gift shop that sells candles, cards, and clocks. The color options for each type of gift are gold, green, or rose. What is the minimum number of shoppers who each purchase one item needed to guarantee that at least three shoppers buy the same item in the same color?
  - Permutations and combinations
  - Binomial theorem

    - **Example problem**: How many distinct rearrangements of the 8 letters ABCDEFGH contain the string FADE?
- **Discrete Probability**
    - Basics, probability of events from counting
    - Properties of probability distributions, computing distributions
    - Law of Total Probability
    - Bayes' Theorem
    - Bayesian Spam Filters
    - Independence/dependence, conditional independence: we saw a number of different tests for independence
    - **Example problem**: Consider rolling two 6-sided dice. One of them is a fair die. The other is unfair, where the numbers 1-4 are all equally likely to be rolled, but the number 5 is twice as likely as the number 1 to be rolled. What are these values from the probability distribution for the set of possible outcomes for the sum of the two dice? Are the two events independent?
- **Recurrence Relations**
    - Solving recurrences:
        - Recursive-plugging-back-in (also known as iteration or unrolling)
        - Combining the homogeneous and particular nonhomogeneous solutions
            - Characteristic polynomial
            - What are our "usual" guess(es) for the form of the particular solution?
            - What is the "usual" guess for the homogeneous solution?
            - What do we do in the repeated roots case? What about if part of the homogeneous solution ends up being part of our guess for the particular solution?
            - How do we solve for the constants? When do we solve for the constants?
    - Counting using recurrence relations
    - Solving counting problems using recurrences vs using dynamic programming
    - **Example problem**: In the kingdom of Hyrule, currency comes in denominations of 1, 2, and 5 rupees. After purchasing a pair of iron boots from a shop, you are owed 17 rupees in change. If the shopkeeper places each gem on the counter one at a time, how many ways are there for him to make your change?

- **Relations**
    - Relations are like functions, but more flexible. We can use relations to model more things than functions alone would allow.
    - Equivalence relations, equivalence classes, partitioning of sets
    - **Example problem**: Determine whether each of the following relations $R \subseteq AxA$, where A is the set of all CU students, is reflexive, symmetric, transitive, and/or an equivalence relation. Briefly justify each conclusion.
        - (a) $(a, b) \in R$ if and only if a shares at least one class with b
        - (b) $(a, b) \in R$ if and only if a has a higher GPA than b
        - (c) $(a, b) \in R$ if and only if a is roommates with b

- **Graphs**
    - Degree of vertices (undirected), in/out-degree (directed)
    - Walks, paths, Eulerian circuits
    - Graph coloring and scheduling problems; chromatic number
    - **Example problem**: For each of the graphs below, determine if the graph has an Eulerian Circuit. If it does, give one such tour. If it does not, explain why.



What should I be doing to prepare?

1. ACTUALLY DO PROBLEMS. Solve them all the way out by hand. This is what you will need to do during the exam.
2. Review the homework and exam problems from this semester. Reattempt tricky ones, and make sure you understand how to do any problems you couldn't before.
    a. HW Solutions are posted on Moodle. Exam solutions are posted on Piazza under resources.
3. Review old exams from previous semester. Solutions are mostly available on Piazza.
4. Review the Workgroup worksheets (Moodle)
5. Read the slides.
    a. Can you do the examples from lecture?
6. Use Piazza for constructive discussion.
    a. The problems you're studying to prepare for the final exam aren't graded work. We strongly encourage you to discuss the problems with each other via Piazza! If you're stumped, that means you've found a nice, challenging problem and others would probably benefit from studying it.
7. Look at the textbook for additional problems
8. Look at the "All Moodle HW and Quiz" problem sets (Moodle)
9. Use this guide as a checklist, not as your only study tool.

Related to the exam:

10. You may use a calculator, provided it cannot access the internet. You may NOT use a smartphone as a calculator.
11. You may bring 1 page of handwritten notes (8.5x11 inches) to the exam.
12. Format: part multiple choice, part short answer (with some justification), part free response (like the written homework problems; extended justification)

You will not need to write any code or use a computer for the exams. You may need to interpret code, fill in some blank lines of code by hand, or deduce what will be the output of Python code.