Greedy Algorithms

- **Write a proof of correctness for a greedy algorithm, following the format from class.**
  - Define a *partial solution* and an *optimum solution* to the given problem.
  - Define what it means for an optimum solution to *extend* a partial solution.
  - Apply the correct proof structure:
    - Proof by induction that every partial solution can be extended to an optimal solution.
    - In the inductive step, top-level cases determined by the conditions in the body of the algorithm's main loop.
    - In the inductive step, sub-cases for each case determined by potential differences between the partial solution and an optimum solution.
- **Write an algorithm that uses a greedy strategy.**

Dynamic Programming

- **Write an algorithm that uses dynamic programming, following the structure from class.**
  - Define an *optimum solution* to the given problem.
  - Describe the recursive structure of optimum solutions by relating them to optimal solutions of subproblems.
  - Define an array that stores the optimum value of a solution to the subproblem defined by the array indices.
  - Give a recurrence relation for the array values based on the recursive structure.
  - Write an iterative algorithm to compute array values "bottom-up," following the recurrence.
  - Use an additional array if needed to reconstruct an optimum solution from the computed array values.
- **Argue the correctness of a dynamic programming algorithm, based on the recursive structure of the problem it solves. Analyze its runtime.**

Graph Algorithms

- **Solve a problem using graph techniques, and argue the correctness of your solution.**
  - Describe how to construct a graph from a problem input.
  - Describe how to reconstruct a solution from a shortest path or minimum spanning tree in the graph.
  - Know algorithms like Dijkstra, Bellman-Ford, Floyd-Warshall, Prim and Kruskal, and how to use them.

Network Flows

- **Solve a problem using network flow techniques, and argue the correctness of your solution.**
  - Describe how to construct a network from a problem input.
  - Describe how to reconstruct a solution from a maximum flow or minimum cut in the network.

  - Argue that every solution to the original problem becomes a valid flow (or cut) in the network, so the maximum flow value (or minimum cut capacity) is at least as good as the optimum solution value.
  - Argue that every valid flow (or cut) in the network becomes a solution to the original problem, so the optimum solution value is at least as good as the maximum flow value (or minimum cut capacity).
  - Know algorithms like Ford-Fulkerson and Edmonds-Karp and how to use them.
- **Prove properties of flows and cuts in networks.**
  - Understand how to construct and use a residual network.
  - Understand the Max-Flow-Min-Cut Theorem and how to use it.

Linear Programming

- **Solve a problem using linear programming, and argue the correctness of your solution.**
  - Describe how to construct a linear program from a problem input: define variables, objective function, and constraints explicitly.
  - Describe how to reconstruct a solution to the original problem from a solution to the linear program.
  - Argue that every solution to the original problem becomes a feasible solution in the linear program, so the optimum value of the objective function is at least as good as the optimum solution value.
  - Argue that every feasible solution in the linear program becomes a solution to the original problem, so the optimum solution value is at least as good as the optimum value of the objective function.
- **Solve a given LPP using the Simplex algorithm.**

P, *NP*, *coNP* and $\leqslant_p$

- **Prove $Y \leqslant_p X$ for specific decision problems X, Y.**
  - Describe an explicit construction f that returns an input $f(y)$ to problem X given an arbitrary input y to problem Y.
  - Argue that the construction f can be carried out in polytime, and $y \in Y \Leftrightarrow f(y) \in X$.
  - Avoid common mistakes:
    * constructions that do **not** run in polytime because they attempt to use a certificate (not part of input y);
    * giving two "half" constructions (constructing $f(y)$ from y and separately constructing $f(x)$ from x);
    * confusing $\leqslant_p$ with the weaker notion of trying to describe an algorithm/verifier for Y that makes calls to an algorithm/verifier for X.
- **Know the definitions of each complexity class (P, NP, $coNP$) and of $\leqslant_p$.**

NP-Completeness

- **Show that a problem *A* is *NP*-complete.**

  - Give a *verifier* for A (**not** a "generate-and-verify" algorithm), and argue that it returns True for *some* certificate iff the input is a yes-instance.
  - Find a suitable NP-hard problem B and prove $B \leqslant_p A$.
  - Avoid the common mistake of trying to show $A \leqslant_p B$.

Self-Reducibility

- **Show that a problem is polytime self-reducible.**
  - Assume the existence of a polytime algorithm DA for the decision problem.
  - Write an explicit algorithm A for the search/optimization problem, making calls to DA.
  - Argue the correctness of algorithm A (usually through an appropriate loop invariant).
  - Analyse the runtime of algorithm A to show that it runs in polytime.

Approximation Algorithms

- **Prove bounds on the approximation ratio for both mini- and maximization problems.**
  - Know the definition of *approximation ratio* for both kinds of optimization problems.
  - With some guidance/hints, prove bounds on the approximation ratio.

In General...

- Pay particular attention to the *marking scheme* for each test and homework: these show you explicitly what aspects of your answers are important!
- Don't forget the tutorial problems: they often contain important clarifications or variations on the ideas presented in lectures.
- *Read the course information sheet* (for details of the course policies, particularly the "20% rule" and information about aid sheets).
- Keep in mind the following guidelines for writing the exam.
  - **Plan your time!** Figure out how many minutes you have available for each page/question/mark.
  - **Read the questions carefully!** If something is unclear, please **ask**.
  - **Show what you know!**
    - Read **every** question before doing anything else.
    - Answer the "easy" questions first.
    - For all questions start with an outline of what you have to do—**this is worth marks**.
    - Go back to the questions you're not sure about and work on them, but keep track of your time.
  - **Explain what you're doing!** A correct ouline of a solution is worth marks.
  - **Don't ramble!** Write concise, to-the-point answers. Incorrect solution elements will cost you marks.

    On the other hand, admit when something does not work: you will get more if we see that you understand your mistakes.
  - (This is the hardest one.) **Relax!** You'll function much better if you are well-rested and relaxed than if you are tired or tense.

It was a pleasure to teach you this term. Good luck on all your exams. Have a great summer!