CS256: Guide to Greedy Algorithms

Based on handouts by Roughgarden, Sharp, Wexler and Schwarz

Greedy algorithms are often simple and intuitive, but can be the hardest algorithms to recognize and analyze as optimal. You can stumble on the right algorithm but not recognize that you have found it, or might find an algorithm you're sure is correct and hit a brick wall trying to formally prove its correctness. This handout discusses how to structure the two major proof techniques we have covered for greedy algorithms.

Format of proofs. Greedy algorithms are often used to solve optimization problems: you want to maximize or minimize some quantity subject to a set of constraints.

When you are trying to write a proof that shows that a greedy algorithm is correct, there are two parts: first, showing that the algorithm produces a feasible solution, and second, showing that your algorithm produces an *optimal* solution, a solution that maximizes or minimizes the appropriate quantity. Feasibility usually (but not always) follows directly from the design and we focus on proving optimality.

Greedy Stays Ahead. One of the simplest methods for showing that a greedy algorithm is correct is to use a "greedy stays ahead" argument. This style of proof works by showing that, according to some measure, the greedy algorithm always is at least as far ahead as the optimal solution during each iteration of the algorithm. Once you have established this, you can then use this fact to show that the greedy algorithm must be optimal.

Typically, you structure a "greedy stays ahead" argument in four steps:

- **Define the solutions.** The greedy will produce some solution G that you will probably compare against some optimal solution $\mathcal{O}$. Introduce some variables describing G and $\mathcal{O}$.
- **Define the stay-ahead measure.** Find the measure you will use to compare G and $\mathcal{O}$. This is often the metric the algorithm is being greedy about.
- **Prove that greedy stays ahead.** Use your measure and show that using it, greedy's solution never falls behind of the optimal solution.
- **Prove optimality.** Using the fact that greedy stays ahead, prove that the greedy algorithm must produce an optimal solution. This argument is often done by contradiction by assuming the greedy solution isn't optimal and using the fact that greedy stays ahead to derive a contradiction.

When writing up a proof of this form, you don't need to explicitly enumerate these steps (we didn't do this in lecture). However, these steps likely need to be here. For example, if you don't prove how the fact that greedy stays ahead implies optimality, you haven't actually proven what you need to prove. Finally, although the name of the argument is "greedy stays ahead," technically you are showing that "greedy never falls behind." That is, you want to show that the greedy solution is at least as good as the optimal solution.

Exchange Arguments Exchange arguments are a powerful and versatile technique for proving optimality of greedy algorithms. They work by showing that you can iteratively transform any optimal solution into the solution produced by the greedy algorithm without changing the cost of the optimal solution.
Typically, exchange arguments are set up as follows:

- **Define the solutions.** You are comparing a greedy solution G to an optimal solution $\mathcal{O}$, so it's best to define these variables explicitly.
- **Compare solutions.** Next, show that if $G \neq \mathcal{O}$, then they must differ in some way. Define this point of departure to use in the rest of the proof.
- **Exchange moves.** Show how to transform $\mathcal{O}$ by exchanging some move by $\mathcal{O}$ for some move by G. Then, prove that in by doing so, you did not increase/decrease the cost of G.
- **Iterate.** Argue that you have decreased the number of differences between G and $\mathcal{O}$ by performing the exchange, and that by iterating this exchange you can turn $\mathcal{O}$ into G without impacting the quality of the solution. Therefore, X must be optimal. (To be very rigorous here, we should prove this by induction, but for the purposes of this class it is okay to just informally explain that iteratively proceeding works.)

Summary. There is no hard and fast rule as to when each should be used. In some cases, exchange arguments can be significantly easier than "greedy stays ahead." Greedy stays ahead is often invoked when there is a clear metric in the problem statement that one can use for it. Exchange arguments are often invoked when all optimal solutions share a nice structure that can be exploited to turn one into the other.