Guide to Greedy Algorithms

Based on a handout by Tim Roughgarden, Alexa Sharp, and Tom Wexler

Greedy algorithms can be some of the simplest algorithms to implement, but they're often among the hardest algorithms to design and analyze. You can often stumble on the right algorithm but not recognize that you've 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've covered for greedy algorithms ("greedy stays ahead" and exchange arguments) and gives some intuition for when one might be appropriate over the other. We recommend referring to the lecture slides for examples of formal, worked examples of these proofs in practice.

Format for Correctness Proofs

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

- Maximize the number of events you can attend, but do not attend any overlapping events.
- Minimize the number of jumps required to cross the pond, but do not fall into the water.
- Minimize the cost of all edges chosen, but do not disconnect the graph.

When you are trying to write a proof that shows that a greedy algorithm is correct, you often need to show two different results. First, you need to show that your algorithm produces a *feasible solution*, a solution to the problem that obeys the constraints. For example, when discussing the frog jumping problem, we needed to prove that the series of jumps the greedy algorithm found actually gave a legal path across the pond. Next, you need to show that your algorithm produces an *optimal solution*, a solution that maximizes or minimizes the appropriate quantity.

It's usually much easier to prove feasibility than to prove optimality, and in lecture we've routinely hand-waved our way through this. When writing up a formal proof of correctness, though, you shouldn't skip this step. Typically, these proofs work by induction, showing that at each step, the greedy choice does not violate the constraints and that the algorithm terminates with a correct solution.

As an example, here is a formal proof of feasibility for Prim's algorithm. In lecture, we saw that the general idea behind the proof was to show that the set T of edges added to the MST so far form a spanning tree of the set S. The proof uses the fact that a tree is a connected graph where $|E| = |V| - 1$ to show that after we add an edge crossing the cut $(S, V - S)$, the set T must be a tree because it has the right number of edges and is still connected.

Here is how we might formalize this:

Theorem (Feasibility): Prim's algorithm returns a spanning tree.

Proof: We prove by induction that after k edges are added to T, that T forms a spanning tree of S. As a base case, after 0 edges are added, T is empty and S is the single node $\{v\}$. Also, the set S is connected by the edges in T because v is connected to itself by any set of edges. Therefore, T connects S and satisfies $|T| = |S| - 1$, so T is a spanning tree of S.

For the inductive step, assume the claim is true after k edges are added. If at this point $S = V$, the algorithm terminates, and since T is a spanning tree of S, T is a spanning tree of V, as required. Otherwise, $S \neq V$, so the algorithm proceeds for another iteration. Prim's algorithm selects an edge (u, v) crossing the cut $(S, V - S)$ and then sets S to $S \cup \{v\}$ and T to $T \cup \{(u, v)\}$ Since at the start of the iteration T was a spanning tree for S, it connected all nodes in S. Therefore, all nodes in S are still connected to one another, and v is now connected to all nodes in $S \cup \{v\}$, since it is connected to u and u is connected to all nodes in S. Finally, note $|T \cup \{(u, v\}| = |S| - 1 + 1 = |S| = |S \cup \{v\}| - 1$, so $T \cup \{(u, v)\}$ is a spanning tree for $S \cup \{v\}$, so the claim holds after $k + 1$ edges have been added. ■

"Greedy Stays Ahead" Arguments

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 would structure a "greedy stays ahead" argument in four steps:

- **Define Your Solution**. Your algorithm will produce some object X and you will probably compare it against some optimal solution X^*. Introduce some variables denoting your algorithm's solution and the optimal solution.
- **Define Your Measure**. Your goal is to find a series of measurements you can make of your solution and the optimal solution. Define some series of measures $m_1(X)$, $m_2(X)$, …, $m_n(X)$ such that $m_1(X^*)$, $m_2(X^*)$, …, $m_k(X^*)$ is also defined for some choices of m and n. Note that there might be a different number of measures for X and X^*, since you can't assume at this point that X is optimal.
- **Prove Greedy Stays Ahead**. Prove that $m_i(X) \geq m_i(X^*)$ or that $m_i(X) \leq m_i(X^*)$, whichever is appropriate, for all reasonable values of i. This argument is usually done inductively.
- **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, if you'll recall). However, these steps likely need to be here. If you don't define your solution and the optimal solution, the notation won't make sense later on. If you don't specify your measurements, you can't prove anything about them. If you forget to prove that greedy stays ahead, the rest of your proof can't assume it. Finally, if you don't prove how the fact that greedy stays ahead implies optimality, you haven't actually proven what you need to prove (namely, that you get an optimal solution.)

The main challenge with this style of argument is finding the right measurements to make. If you pick the wrong measurements, you will either get stuck proving that greedy stays ahead (often because it doesn't always stay ahead!) or proving that because greedy stays ahead, the algorithm must be optimal (often because you're using the wrong measurement). As mentioned in lecture, it's often useful to try showing that *if* greedy stays ahead according to your measurements, then the algorithm is optimal before you try actually showing that greedy stays ahead. That way, you won't end up trying to prove that greedy stays ahead in a measurement with no bearing on the result.

A few other common pitfalls to watch out for:

- When defining your measurements, make sure that you define them in a way that lets you measure the optimal solution you're comparing against, especially if that solution wasn't generated by the greedy algorithm. It's typically easy to find measurements of the greedy solution because it's generated one step at a time; the challenge is figuring out how to determine what specifically those measurements are made on. For example, in the interval scheduling problem, the measurements made corresponded to the end times of the events as they were added to the greedy solution. To make those measurements applicable to the arbitrarily-chosen optimal schedule S^*, we had to define those measurements on an absolute scale: our measurements measured the finishing times of the ith event to finish, sorted by order of finishing time.
- Be wary of the case where the greedy solution and optimal solutions don't necessarily have the same sizes as one another. In some problems, such as the MST problem, all legal spanning trees have the same sizes as one another, though they have different costs. In others, such as a the frog jumping problem, different solutions might have different numbers of jumps in them. If you do use a "greedy stays ahead" argument, you should be sure that you don't try showing that your greedy algorithm is better by some measure m_n if the optimal solution can only be measured by $k < n$ measurements.
- Although the name of the argument is "greedy stays ahead," you are usually more properly 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, not *strictly better than* the optimal solution.

It takes some practice to know when a "greedy stays ahead" proof will be appropriate. Often, these arguments are useful when optimal solutions might have different sizes, since you can use the fact that greedy stays ahead to explain why your solution must be no bigger / no smaller than the optimal solution: if the greedy algorithm did/didn't terminate at some step, then the optimal solution at that point must be too big / too small and therefore is incorrect. However, there's no hard-and-fast rule about where to apply this proof technique, and you'll build experience working with it as you work through the problem set on greedy algorithms.

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, thereby proving that the greedy solution is optimal.

Typically, exchange arguments are set up as follows:

- **Define Your Solutions**. You will be comparing your greedy solution X to an optimal solution X^*, so it's best to define these variables explicitly.
- **Compare Solutions**. Next, show that if $X \neq X^*$, then they must differ in some way. This could mean that there's a piece of X that's not in X^*, or that two elements of X that are in a different order in X^*, etc. You might want to give those pieces names.
- **Exchange Pieces**. Show how to transform X^* by exchanging some piece of X^* for some piece of X. You'll typically use the piece you described in the previous step. Then, prove that by doing so, you did not increase/decrease the cost of X^* and therefore have a different optimal solution. (You might also be able to immediately conclude that you've strictly worsened X^*, in which case you're done. This is uncommon and usually only works if there's just one optimal solution.)
- **Iterate**. Argue that you have decreased the number of differences between X and X^* by performing the exchange, and that by iterating this process you can turn X^* into X without impacting the quality of the solution. Therefore, X must be optimal. (To be very rigorous here, you would probably proceed by induction to show this, but for the purposes of this class it's fine to hand-wave and explain why you could iteratively transform the solution.)

In some cases, exchange arguments can be significantly easier than "greedy stays ahead" arguments, since often the reason you were greedily optimizing over some quantity is because any solution that doesn't greedily optimize that quantity is either suboptimal or could be locally modified to optimize it without changing its cost.

When writing up exchange arguments, watch out for the following:

- Your solution X could be optimal even if $X \neq X^*$, since there can be many optimal solutions to the same problem. Approaching these proofs by contradiction can get you stuck because you won't actually get a contradiction by making a local modification to the optimal solution. Typically, you will only get a contradiction if there is only one optimal solution.
- Your solution may need to iteratively transform X^* into X. Making one modification makes X^* and X closer to one another, but it doesn't guarantee that X and X^* are now equal. This is why we suggest ending your proof with an iteration argument explaining why the procedure can be iterated.
- Be sure your proof accounts for why the piece of X you're exchanging for a piece of X^* actually must exist in the first place. This shouldn't be too hard, since it typically follows from the fact that $X \neq X^*$, but you should offer some justification.

As with "greedy stays ahead," there is no hard-and-fast rule explaining when this proof technique will work. Exchange arguments work particularly well when all optimal solutions have the same size and differ only in their cost, since that way you can justify how you are going to remove some part of the optimal solution and snap another one in its place. Determining when to use exchange arguments is a skill you'll pick up as you practice on the problem set.