Greedy algorithms for combinatorial optimization typically share a common structure. It helps to keep this structure in mind when you are trying to design and prove correctness for your own algorithms.

1 The Setting: Combinatorial Optimization

A combinatorial optimization problem requires us to identify an optimal feasible solution to a problem instance. Feasibility is defined by a set of combinatorial constraints that any feasible solution must satisfy. Solutions are ranked by an objective, or measure of quality; an optimal feasible solution maximizes this measure among all feasible solutions. (Equivalently, it may minimize some measure of badness.)

As a simple example, consider the following scheduling problem. A problem instance is a set $P$ of jobs $p_i$, with each job occupying a fixed time interval $[s_i, s_i + \ell_i)$. The goal is to schedule the maximum possible number of non-overlapping jobs. A solution $\pi$ to $P$ is any subset of jobs from $P$. A feasible solution is one in which no two jobs overlap in time, while an optimal feasible solution also maximizes the number of jobs scheduled, i.e. the size $|\pi|$.

2 What is a Greedy Algorithm?

Let $P$ be an instance of a combinatorial optimization problem. We will consider algorithms that incrementally build up a solution to $P$ by making a sequence of choices. These choices are not independent – a choice made early in the algorithm might restrict the range of choices that can be made later on without violating a feasibility constraint. For example, in our scheduling problem, adding a job $p_i$ to the solution prevents us from later adding other jobs that overlap $p_i$ in time.

In general, we might have to be very careful to avoid making a choice that is incompatible with optimality. For example, if job $p_i$ happens to be present in every optimal solution, then choosing another job $p_j$ that overlaps it is guaranteed to lead to a suboptimal solution. If we can’t predict the consequences of our choices, then we may be stuck using an expensive trial-and-error search of the solution space to find an optimum!

Fortunately, in some problems, there is a choice criterion – a rule for making a choice at each step that is guaranteed to eventually produce an optimal solution. This criterion typically involves making a feasible choice that minimizes or maximizes some local measure of goodness. Because we make the locally best choice at each step, this choice is called a greedy choice, and the resulting algorithm is called a greedy algorithm.

One greedy algorithm for our scheduling problem is the following. Define the finishing time $f_i$ of job $p_i$ to be $s_i + \ell_i$. Repeatedly add to the solution the job with the earliest finishing time that does not overlap with any previously chosen job, until no more jobs can be chosen. The italicized directions above constitute the greedy choice, which we simply iterate until we are done.
3 Proving that a Greedy Choice is a Good Choice

We claim that our greedy scheduling algorithm yields an optimal feasible solution to \( P \), i.e. one that maximizes the number of jobs chosen. It’s not hard to see that the solution is feasible, since we never choose a job that overlaps a previously chosen job. But how can we prove that the solution is optimal?

Fortunately, there’s a standard inductive “argument template” for proving optimality of greedy algorithms. Here’s how it works.

1. **To prove**: the greedy algorithm works on problem instances of all sizes.

2. We will proceed by induction on the size of the problem instance \( P \).

3. **Bas**: For the smallest possible \( P \), we show directly that making the greedy choice leads to an optimal solution. (The smallest \( P \) is typically empty or, if that is not allowed, contains only one element.)

4. **Ind**: Let \( P \) be given, and let \( \hat{c} \) be the greedy choice made by the algorithm for \( P \).

5. After making choice \( \hat{c} \), show that we are left with a strictly smaller subproblem \( P' \) of \( P \).

6. By the IH, applying the greedy algorithm to \( P' \) yields an optimal feasible solution \( \pi' \).

7. Finally, we show that combining \( \pi' \) with our greedy choice \( \hat{c} \) yields a feasible solution to \( P \), and that this solution is optimal. QED

Fleshing out this template requires that we provide a base case argument in point 3 (required, but usually easy), and, crucially, that we provide justifications for points 5 and 7. Here follows some advice on how to supply these essential parts of the proof.

7a. **Show that making the greedy choice is always compatible with optimality.** In other words, show that there is always an optimal solution to \( P \) that includes the greedy choice, and so making this choice can never prevent us from finding an optimum. This claim is sometimes called the *greedy choice property* for the algorithm.

Here’s an example of this kind of argument for our scheduling algorithm. Suppose \( \pi^* \) is any optimal solution to problem instance \( P \), and again let \( \hat{p} \) be the greedy choice, i.e. the job in \( P \) with the earliest finishing time. If \( \hat{p} \in \pi^* \), then we are done. Otherwise, we claim that some other optimal solution \( \hat{\pi} \) contains \( \hat{p} \).

Indeed, \( \pi^* \) must contain some other job \( p' \) that overlaps \( \hat{p} \); otherwise, we could add \( \hat{p} \) to \( \pi^* \) and so get a “better-than-optimal” solution, which is impossible. Now \( p' \) finishes at or after the finishing time of \( \hat{p} \), so every job that overlaps \( \hat{p} \) overlaps \( p' \) as well. Hence, dropping \( p' \) from \( \pi^* \) and adding \( \hat{p} \) results in a new feasible solution \( \hat{\pi} \) that is just as good – and therefore still optimal. QED

*(Note 1: this kind of argument is often called an *exchange* because it involves eliminating a choice from an optimal solution and replacing it with the greedy choice. The key is to show that the replacement results in a solution that is (a) feasible and (b) at least as good as the original.)*
(Note 2: sometimes, the greedy choice may be the only choice that leads to optimality. In this case, if you assume that some optimum \( \pi^* \) fails to include the greedy choice, you wind up with a contradiction. This is fine – you’re trying to show that there exists an optimum that makes the greedy choice, and you’ve actually proven that all such optima do so!)

5. and 7b. Show that making the greedy choice leaves a strictly smaller subproblem, such that any feasible solution to this subproblem can be combined with the greedy choice to yield a feasible solution to the original. I refer to this claim as the inductive structure property for the algorithm. It usually doesn’t occupy a lot of words in your proof; rather, it “falls out” as a consequence of correctly defining your algorithm in the first place. However, you need to acknowledge in your proof that you checked this condition and explain why it is true if it is not obvious.

Another way to state the inductive structure property is that making the greedy choice leaves no external constraints on the subproblem that could make an arbitrary feasible solution to it unusable in the original problem.

In our scheduling example, it’s not clear that our algorithm as written satisfies the “no external constraints” criterion. Indeed, after we choose our first job, say \( \hat{p} \), simply repeating the algorithm on the remaining jobs \( P' = P - \hat{p} \) might pick a solution that includes one of the jobs that overlaps with \( \hat{p} \)! We could not combine such a solution with \( \hat{p} \) to solve \( P \), since the result would not be feasible.

To fix this issue, we can restate our algorithm more carefully. Once we choose job \( \hat{p} \) according to our greedy criterion, we create the subproblem \( P' \) by removing from \( P \) both \( \hat{p} \) and any jobs that overlap \( \hat{p} \). Now, no job of \( P' \) can overlap \( \hat{p} \), so it is always feasible to combine any solution to \( P' \) with \( \hat{p} \).

7c. Show that combining the greedy choice with an optimal subproblem solution yields an optimal solution to \( P \). This claim is sometimes called the optimal substructure property for the algorithm. Its proof usually takes the form of a contradiction argument, in which the greedy choice property we proved above plays an essential role.

In our scheduling problem, suppose that removing the greedy choice \( \hat{p} \) and its overlapping jobs from \( P \) leaves the subproblem \( P' \), which has an optimal solution \( \pi' \). Denote by \( V(\pi) \) the objective value associated with a solution \( \pi \), i.e. the size \( |\pi| \).

Now suppose that \( \pi = \pi' \cup \{\hat{p}\} \) is not an optimal solution to \( P \). Let \( \pi^* \) be an optimal solution to \( P \) that contains the greedy choice \( \hat{p} \) (which is known to exist by the greedy choice property). If we remove \( \hat{p} \) from \( \pi^* \), the remaining sub-solution is a feasible solution to \( P' \) and has objective value

\[
V(\pi^* - \{\hat{p}\}) = V(\pi^*) - 1.
\]

But \( \pi' \), which is also a solution to \( P' \), has objective value

\[
V(\pi') = V(\pi - \{\hat{p}\}) = V(\pi) - 1 < V(\pi^*) - 1,
\]

which contradicts the optimality of \( \pi' \). Conclude that \( \pi \) must in fact be optimal for \( P \). QED

(Note: this argument works without modification for any objective \( V \) that can be decomposed into a sum or product of two parts, one of which depends only on the greedy choice and the other of which depends only on the remaining subproblem. A common case is that \( V(\pi \cup \{p\}) = V(\pi) + f(p) \) for some function \( f \)).
4 Advice for Your Homework

To prove optimality of a greedy algorithm for class purposes, it is enough to prove the greedy choice, inductive structure, and optimal substructure properties for the algorithm. Please clearly divide your argument into the above three pieces, with suitable headings. You need not write down the full inductive template, though it may help you to do so the first couple of times to make sure you understand your proof obligations.

Initially, I would advise that you always write out the full contradiction proof for the optimal substructure property. Once you are more comfortable, you can “shortcut” this step of the proof by showing that the objective has the form $V(\pi) + f(p)$, as suggested above, then saying “apply the standard contradiction argument.” Don’t do this until you are sure you are justified!