Main Steps

There are four main steps for a greedy stays ahead proof.

Step 1: Define your solutions. Describe the form your greedy solution takes, and what form some other solution takes (possibly the optimal solution). For example, let A be the solution constructed by the greedy algorithm, and let O be a (possibly optimal) solution.

Step 2: Find a measure. Find a *measure* by which greedy stays ahead of the other solution you chose to compare with. Let $a_1, \ldots, a_k$ be the first k measures of the greedy algorithm, and let $o_1, \ldots, o_m$ be the first m measures of the other solution ($m = k$ sometimes).

Step 3: Prove greedy stays ahead. Show that the partial solutions constructed by greedy are always just as good as the initial segments of your other solution, based on the measure you selected.

- For all indices $r \leq \min(k, m)$, prove (often by induction) that $a_r \geq o_r$ or that $a_r \leq o_r$, whichever the case may be. Don't forget to use your algorithm to help you argue the inductive step.

Step 4: Prove optimality. Prove that since greedy stays ahead of the other solution with respect to the measure you selected, then it is optimal.

Comments

- The tricky part is finding the right measure; greedy won't necessarily stay ahead in just any measure.

- Make sure your measure guarantees greedy is optimal, that is, if greedy stays ahead with respect to this measure, how does it guarantee your greedy solution is optimal?

- If the problem you are solving is not an optimization problem, but rather, is a yes/no question, then you will have to modify the general structure to not talk about "optimal", but rather, any arbitrary "yes" solution. That is, given an arbitrary "yes" instance, you need to show that your algorithm indeed returns "yes". This will require greedy stays ahead. You also would need to argue that your algorithm returns "no" when given a "no" instance; usually this is easier to argue directly from the nature of your algorithm.

Example: Interval Scheduling

Suppose you have a set of n requests $\{1, 2, \ldots, n\}$, each with a desired start and finish time pair (s_i, f_i). We determine a schedule with the maximum number of non-overlapping (compatible) requests by repeatedly selecting from the remaining request the one with the earliest finish time, and

removing all conflicting requests from the set. We will prove this returns an optimal solution.

Let $A = \{i_1, \ldots, i_k\}$ be the set of requests selected by our greedy algorithm, in the order in which they were added. Let $O = \{j_1, \ldots, j_m\}$ be the requests selected by an optimal solution, ordered by their finish times.

We will compare A and O by their jobs' finish times, that is, we define the measures $a_r = f(i_r)$ and $o_r = f(j_r)$ for all $r \leq k$, and we show that for all $r \leq k$, $a_r \leq o_r$ (i.e. that $f(i_r) \leq f(j_r)$). This can be shown by induction on r.

Formally, for all $r \leq k$, let $P(r)$ be the statement that $a_r \leq o_r$. We want to show that $P(r)$ is true for all $1 \leq r \leq k$.

As a base case, consider when $r = 1$. Since the algorithm selects the job with the earliest finish time, it certainly must be the case that $a_1 = f(i_1) \leq f(j_1) = o_1$.

For the inductive hypothesis, suppose that $P(r-1)$ were true for some fixed $r > 1$, that is, suppose that $a_{r-1} = f(i_{r-1}) \leq f(j_{r-1}) = o_{r-1}$.

Now we prove that $P(r)$ is true, using the inductive assumption that $P(r-1)$ is true. That is, we prove that $a_r \leq o_r$. Recall that by the inductive hypothesis, $f(i_{r-1}) \leq f(j_{r-1})$, and so any jobs that are compatible with the first $r-1$ jobs in the optimal solutions are certainly compatible with the first $r-1$ jobs of our greedy solution. Therefore, we could add j_r to our greedy solution, and since we take the compatible job with the smallest finish time, it must be the case that $f(i_r) \leq f(j_r)$, that is, that $a_r \leq o_r$, as desired.

Thus we have shown that for all $r \leq k$, $f(i_r) \leq f(j_r)$. In particular, $f(i_k) \leq f(j_k)$. If A is not optimal, then it must be the case that $m > k$, and so there is a job j_{k+1} in O that is not in A. This job must start after O's k^{th} job finishes at time $f(j_k)$, and hence after $f(i_k)$. But then this job is compatible with all the jobs in A, and so A would have added it during the greedy algorithm. This is a contradiction, and thus A has as many elements as O.

The algorithm begins by sorting the n requests in order of finishing time, which takes time $O(n \log n)$. Each time we select an interval, we proceed past any incompatible intervals in our list; that is, we proceed through our list exactly once. This part of the algorithm takes time $O(n)$; therefore, the total running time is $O(n \log n)$.