CS 482 Summer 2003

Proof Techniques: Greedy Stays Ahead

Tom Wexler Alexa Sharp

Greedy algorithms are generally of the form: select a candidate via a greedy manner, and add it to the solution if it doesn't corrupt feasibility. Repeat until finished. "Greedy stays ahead" is one of the techniques used in proving the correctness of greedy algorithms. The idea of a greedy stays ahead proof is to inductively prove that under some measure, the partial solutions produced by the greedy algorithm "stay ahead" of the partial solutions produced by any other algorithm. Generally such a proof then goes on to show that since the final solution produced by the greedy algorithm is ahead of any other solution, the greedy algorithm does in fact return the optimal solution.

Main Steps

After describing your algorithm, the 4 main steps for a "greedy stays ahead" proof are as follows:

Step 1: Label your algorithm's partial solutions, and a general solution. For example, let $A = \{a_1, a_2, \ldots, a_k\}$ be the solution generated by your algorithm, with elements appearing in the order they are added. Let $O = \{o_1, o_2, \ldots, o_m\}$ be an arbitrary (or optimal) feasible solution. Note that while A has a natural ordering, it is no longer obvious how we should order the elements of O. The ordering of O you choose should be picked with your measure and the remainder of the proof in mind. Usually the selection of your measure and the ordering of O go hand in hand.

Step 2: Find a measure. Define a *measure* $f(\cdot)$ by which greedy stays ahead of a general/optimal solution. For example, $f(a_1, \ldots, a_j)$ might be the last completion time of the first j jobs, the total weight of the first j nodes, or the largest index of the first j characters. Of course, $f(o_1, \ldots, o_j)$ should also be well defined.

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 $r \leq k$, prove by induction that $f(a_1, \ldots, a_r)[\geq, \leq]f(o_1, \ldots, o_r)$. 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, the final output is optimal.

Comments

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

Example: Interval Scheduling

Suppose you have a set of n requests $\{1, 2, \ldots, n\}$, and each request i has 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 the remaining request 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 requests selected by our greedy algorithm, in the order in which they were added. Let $O = \{j_1, \ldots, j_m\}$ be an optimal solution, ordered by finish times.

Let $f(S)$ be the last finish time of the jobs in a set S. Note that by our orderings of A and O, $f(i_1, \ldots, i_r) = f(i_r) = f_{i_r}$ and $f(j_1, \ldots, j_r) = f(j_r) = f_{j_r}$, so we will use these interchangeably in this example. Now our goal is to show that for all $r \leq k$, $f(i_r) \leq f(j_r)$.

This can be shown by induction. As the base case, we take $r = 1$. Since we selected the job with the earliest finish time, it certainly must be the case that $f(i_1) \leq f(j_1)$.

For $t > 1$, assume the statement is true for $t - 1$ and we will prove it for t. The induction hypothesis states that $f(i_{t-1}) \leq f(j_{t-1})$, and so any jobs that are valid to add to the optimal solution are certainly valid to add to our greedy solution. Therefore, it must be the case that $f(i_t) \leq f(j_t)$.

So we have 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 $f(j_k)$ (i.e. $s_{j_{k+1}} > f(j_k)$), and hence after $f(i_k)$, by our previous result. But then this job would have been compatible with all the jobs in A, and so our greedy algorithm would have added j_{k+1} to A. This is a contradiction, and thus A is optimal.