CS 482 Summer 2003

Proof Techniques: Greedy Exchange

Tom Wexler Alexa Sharp

Greedy solutions 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 if not finished. "Greedy Exchange" is one of the techniques used in proving the correctness of greedy algorithms. The idea of a greedy exchange proof is to morph a solution produced by an optimal algorithm into the solution produced by your greedy algorithm in a way that doesn't worsen the solution's quality. This shows that any optimal solution is no better than the greedy solution, which proves that greedy does in fact return an optimal solution.

Main Steps

After describing your algorithm, the 3 main steps for a greedy exchange argument proof are as follows:

Step 1: Label your algorithm's solution, and an optimal solution. For example, let $A = \{a_1, a_2, \ldots, a_k\}$ be the solution generated by your algorithm, and let $O = \{o_1, o_2, \ldots, o_k\}$ be an optimal solution.

Step 2: Compare greedy with optimal. Assume that your optimal solution is not the same as your greedy solution. Then either

- there is an element of O not in A and an element of A not in O, or
- there are 2 consecutive elements in O in a different order than they are in A (i.e. there is an inversion).

Step 3: Exchange. Swap or exchange the elements in question in O to make it more similar to A (either swap one element out and another in for the first case, or swap the order of the elements in the second case), and argue that you have a solution that is no worse than before. If we continue swapping, we can eliminate all differences between O and A without worsening the quality of the solution. But then we have shown that greedy is just as good as any optimal solution, and hence is optimal itself.

Comments

- Be careful about using proofs by contradiction starting with the assumption $G \neq O$. Just because your greedy solution is not equal to the selected optimal solution does not mean that greedy is not optimal – there could be many optimal solutions, and your greedy one just isn't the optimal solution you selected. So assuming $G \neq O$ may not get you any contradiction at all, even if greedy works.
- You need to argue why the 2 elements you're swapping even exist out of order, or exist in O but not in A,etc.
- Remember you need to argue that *multiple* swaps can get you from your selected solution to greedy, as one single swap will usually not suffice.
- Keep in mind that all this swapping is part of the proof, not your actual algorithm. You are not describing something you actually do, merely arguing why what you did was ok. On the up side, this means that you don't have to worry about using an inefficient number of swaps, so long as you can show that eventually you manipulate O into A without increasing its cost.

Example: Minimum Spanning Tree

We are given a graph $G = (V, E)$, with costs on the edges, and we want to find a spanning tree of minimum cost. We use Kruskal's algorithm, which sorts the edges in order of increasing cost, and tries to add them in that order, leaving edges out only if they create a cycle with the previously selected edges.

Let $T = (V, F)$ be the spanning tree produced by Kruskal's algorithm, and let $T^* = (V, F^*)$ be a minimum spanning tree. If $F^* \neq F$, and there is an edge $e \in F^*$ such that $e \notin F$. Then e creates a cycle C in the graph $T + e$, and at least one edge f of this cycle crosses the cut defined by $T^* - e$. Furthermore, $e \notin F$ because we tried to add it after the rest of C was already in the tree. Then it was the most expensive edge in C, and so $cost(f) \leq cost(e)$. If we add the edge f to the graph $T^* - e$, then we reconnect the graph and create a spanning tree.

Also, $cost(T^* - e + f) = cost(T^*) - cost(e) + cost(f) \leq cost(T^*)$, and so we have created a new spanning tree of no more cost than T^*, but with one more edge in common with T. We can do this for every edge that differs between T and T^*, until we obtain the tree T, of no more cost than T^*. So $cost(T) \leq cost(T^*)$, and since T^* is a min-cost tree, $cost(T^*) \leq cost(T)$. Together, these imply $cost(T) = cost(T^*)$ and T is min-cost.