

Alternate Methods to Prove Correctness for Greedy Algorithms

September 9, 2004

The text book focuses on a single approach for proving that greedy algorithms are correct. Namely, proving the **greedy choice** and **optimal substructure** properties hold. This is a very useful framework for many problems. However, there are many correct greedy algorithms in which the easiest proof of correctness does not follow this framework.

Here we illustrate some other types of proofs that can be used for proving a greedy algorithm is correct. Of course, these examples are not exhaustive — You may sometimes find a simple direct proof or some other approach to use.

Using a Proof by Contradiction

Here's an example in which a proof by contradiction can be used for the correctness proof. (Note that this is different than the case in which a proof by contradiction is used to show that the greedy choice property holds.)

We consider the problem of optimally arranging files on a tape. You are given n files of lengths l_1, l_2, \dots, l_n . The user requests file i to be retrieved with probability p_i . Hence, the expected cost of retrieving the n files is $T = \sum_{i=1}^n (p_i \cdot L_i)$, where $L_i = \sum_{j=1}^i l_j$ (the total length of all files preceding and including i on the tape). The goal is to determine an ordering of the files that minimizes T .

To solve this problem, we use the following greedy algorithm. Store the files in increasing order of $\frac{l_i}{p_i}$. The time complexity for this algorithm is $O(n \log n)$ with the sorting dominating. We now prove that this greedy solution is correct by using a proof by contradiction to prove that any other file order cannot be optimal. Suppose there is an optimal solution where files are not sorted in the increasing order of l_i/p_i . Then, there must be a *neighboring* pair (a, b) such

$$\text{that } \frac{l_a}{p_a} > \frac{l_b}{p_b} \quad (*)$$

and yet a is stored before b . We show that interchanging their order reduces the expected retrieval cost. Since a, b are neighbors, interchanging their order *does not* affect the retrieval cost for *any other files*. Thus, the only change in the summation defining T is for the terms corresponding to files a and b . The retrieval cost before the swap is: $p_a(L + l_a) + p_b(L + l_a + l_b)$ where L is the total length of files preceding a . After the swap, the cost for accessing a and b is: $p_b(L + l_b) + p_a(L + l_b + l_a)$. Suppose the swap didn't reduce the cost. Then,

$$p_a(L + l_a) + p_b(L + l_a + l_b) \leq p_b(L + l_b) + p_a(L + l_b + l_a)$$

Canceling common terms, we get $p_b l_a \leq p_a l_b$, or $\frac{l_a}{p_a} \leq \frac{l_b}{p_b}$, which contradicts $(*)$. Thus, the swap must reduce the total access cost, and hence the optimal tape storage order must be in the increasing order of $\frac{l_i}{p_i}$.

Using an Invariant

A very good example where the framework of proving the greedy choice and optimal substructure property does not work is for the Single-Source Shortest Path Problem (see pages 514-531 of the text). In this problem, the input is a directed weighted $G = (V, E)$ and a source vertex $s \in V$.

The goal is to find a shortest path from s to each vertex $v \in V - \{s\}$ where the length of path p is $w(p) = \sum_{(u,v) \in p} w(u,v)$. We denote the length of the shortest path from s to v by $\delta(v)$

Dijkstra's algorithm is a greedy algorithm to solve the single source shortest path problem for graphs with no negative-weight edges. The algorithm works as follows. While the shortest paths are being built, a partitioning of the vertices is maintained, S and $V - S$. S is the set of vertices for which the shortest paths (from s) have been obtained already. Initially $S = \emptyset$. Each vertex in V is placed in a priority queue Q . (In general, Q will hold each vertex in $V - S$.) For each vertex $v \in S$ we have a variable $d[v]$ which is the length of the shortest path from s to v that has been found so far. For each vertex $v \in Q$, $d[v]$ is the key. At termination, for all $v \in V$, $d[v]$ is the length of the shortest path from s to v . If $d[v] = \infty$ upon termination then there is no path from s to v . Here's high-level pseudo-code for Dijkstra's algorithm:

```

for each vertex v in V                                // initialization
    d[v] = infinity
d[s] = 0
for each vertex v in V
    insert v into priority queue Q with key d[v] //end of initialization

while Q is not empty                                  //main loop
    u = vertex in Q with min value of d[u]           //place u into S
    for each vertex v reachable from u
        if d[v] > d[u]+w(u,v)                       //better path to v found via u
            d[v] = d[u]+w(u,v)                       //update d[v]

```

We now prove that Dijkstra's algorithm is correct by by proving (by induction on $|S|$) that the following invariant holds: $\forall v \in S, d[v] = \delta(v)$.

Base: Initially $S = \emptyset$ and hence the invariant is trivially true.

Inductive Step: Assume that when $|S| = i$, the invariant holds (i.e. $\forall u \in S, d[u] = \delta[u]$). We now prove that the invariant will be true when $|S| = i + 1$. Let v be the $(i + 1)^{\text{st}}$ vertex extracted from Q (and hence placed in S) and let p be a path from s to v with weight $d[v]$ (which clearly exists and in fact can be easily saved by the algorithm). Let u be the vertex just before v in p . Since paths to vertices in Q are only considered that use vertices from S , $u \in S$ and hence by the inductive hypothesis $d[u] = \delta[u]$.

We now prove that p is a shortest path from s to v . Assume the contrary, i.e., there exist a path p^* from s to v where $w(p^*) = \delta(v) < w(p)$. Since p^* connects a vertex in S to a vertex in $V - S$, there must be a first edge $(a,b) \in p^*$ where $a \in S$ and $b \in V - S$. We can partition the path p^* as $p_1 \cdot (a,b) \cdot p_2$ where p_1 is the portion of p^* from s to a and p_2 is the portion of p^* from b to v and “ \cdot ” is concatenation. By the inductive hypothesis $d[a] = \delta(a)$, Furthermore since p^* is a shortest path, it follows that $p_1 \cdot (a,b)$ must be a shortest path from s to b (a shorter path from s to b would contradict the optimality of p^*). When a was placed in S the edge (a,b) was considered and hence $d[b] = \delta(b)$. Since v was the $(i + 1)^{\text{st}}$ vertex from Q while b was still in Q , it follows that $d[v] \leq d[b]$. Furthermore, since the edge weights are all non-negative $\delta(v) = w(p^*) \geq d[b]$. Combined with $d[v] \leq d[b]$ yields that $w(p^*) \geq d[v] = w(p)$ which contradicts that $w(p^*) < w(p)$ completing the inductive proof.