DS 320 Algorithms for Data Science **Greedy Exchange**

Greedy solutions generally take the following form. Select a candidate greedily according to some heuristic, and add it to your current solution if doing so 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 incrementally modify a solution produced by any other algorithm into the solution produced by your greedy algorithm in a way that doesn't worsen the solution's quality. Thus the quality of your solution is at least as great as that of any other solution. In particular, it is at least as great as an optimal solution, and thus, your algorithm does in fact return an optimal solution.

Main Steps

After describing your algorithm, there are three main steps for a greedy exchange argument proof.

Step 1: Label your algorithm's solution, and a general solution. For example, let $A = \{a_1, a_2, \ldots, a_k\}$ be the solution generated by your algorithm, and let $O = \{o_1, o_2, \ldots, o_m\}$ be an arbitrary (or optimal) feasible solution.

Step 2: Compare greedy with other solution. Assume that your arbitrary/optimal solution is not the same as your greedy solution (since otherwise, you are done). Typically, you can isolate a simple example of this difference, such as one of the following:

- There is an element of O that is not in A and an element of A that is 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 the elements in question in O (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. Then argue that if you continue swapping, you can eliminate all differences between O and A in a finite number of steps without worsening the quality of the solution. Thus, the greedy solution produced is just as good as any optimal (or arbitrary) 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. Also, make sure that any step you make (and not just the first one) doesn't hurt the solution quality.
- Note that the swapping is not part of your algorithm; it is only part of the argument that shows your algorithm is correct.

Example: Flexible Scheduling

We are given a set of tasks, each of which has a duration t_i and a deadline d_i. Our goal is to order the tasks to minimize the lateness of the latest task. Our algorithm will order the tasks from earliest to latest deadline, and complete the tasks in this order.

Proof of Correctness:

Step 1: Label the jobs by deadline, in nondecreasing order. Consider an arbitrary ordering of the jobs. Let $\pi(i)$ be the position of job i in the ordering. (So $\pi(i) = 1$ means job 1 is first.)

Step 2: Assume there is an inversion somewhere in π. That is, there are two jobs i and j such that $i < j$ but $\pi(i) > \pi(j)$.

Step 3: We can find two adjacent jobs i' and j' such that $i < j$ and $\pi(j) \leq \pi(j') < \pi(i') \leq \pi(i)$ by stepping forward in the ordering π starting at j and stopping at the first place where the index of the current job decreases. Swap jobs i' and j'.

We will show that this does not increase the lateness of the ordering π. To see this, we make two observations:

1. The lateness of job i' only decreases from the swap.
2. If job j' is late after the swap, it was less late than job i' before the swap. This follows from the fact that after the swap, j' ends at the same time that i' ends before the swap, but j' has a later deadline.

Consequently, swapping i' and j' can only improve the lateness of the schedule π. Since we may perform this type of swap as long as there is an inversion somewhere in π, repeatedly swapping jobs until there are no further inversions shows that the lateness of the greedy ordering (which has no inversions) is less than or equal to the lateness of the competing ordering π.

The runtime of the greedy algorithm is just $O(n \log n)$—it's just a sort.