Greedy Algorithms

Hello, welcome to competitive programming. There are some problems where you want to maximize some global property, and it so happens that the way to do this is to maximize that property locally. Such algorithms are called Greedy Algorithms.

Objectives

Your objective is to be able to recognize when a greedy algorithm is the correct solution and know how to implement one.

How to Recognize a Greedy Algorithm

A greedy algorithm will have two properties: First, it will have “optimal substructures”, meaning that if you take a subproblem, those subproblems will have optimal solutions.

Second, it has the “greedy property”, meaning that a choice that is optimal locally will also turn out to be optimal globally.

Example

As an example, let’s take the classic coin change problem. We have some coins of certain fixed values and want to count out a specific amount of change using the fewest number of coins.

Depending on the denominations you have, this problem can be greedy or it may need a more powerful algorithm.

An example of the greedy version uses the US coin denominations: 25, 10, 5, and 1 cent.

If I want to give you 57 cents, then I repeatedly take the largest coin value I can until I reach the final value. In this case, I take two 25 cent coins to get 50 cents, leaving 7. To get 7 cents, I take a 5 cent piece leaving 2 cents, which I fill taking two pennies.

Notice that the 7 cent problem is a subproblem of the 57 cent problem, and we solved it optimally. This is an example of the optimal substructure property.

The greedy property is a bit harder to recognize, but the idea is that if you make a greedy decision, you won't regret it later.

As a counter-example, suppose we had a new kind of coin that was worth 20 cents.

If we want to make 40 cents in change, the optimal solution is to use two of the 20 cent coins. But the greedy property no longer holds: if I use a greedy algorithm, we will start by taking a 25 cent piece, and then have to use a 10 cent and a 5 cent piece to make up the difference. Taking the largest possible coin first caused us trouble later. We have to use dynamic programming to solve this problem.

Greedy algorithms in contest

There are two other classical greedy algorithms covered by the text: load balancing and interval covering. You will get problems that illustrate all of these.

In an actual programming contest, the important skill will be knowing for sure if a problem can be solved by a greedy algorithm. Greedy algorithms tend to be fast, but if you are wrong then you will get Wrong Answer verdicts. The general advice then is to use complete search or dynamic programming, because then you are guaranteed to get the right answer.

Next
Transcript — IO →