Greedy Questions

ICT and SCT Captains

December 2020

1 Introduction

A greedy algorithm is one that, as the name suggests, seeks the best immediate choice at any given moment. By continuing to make partially optimal choices, a greedy algorithm will find a globally optimal solution. There should never be an occasion where a greedy algorithm backtracks to fix a previous decision. Greedy questions are often combined with other techniques that are used during the process of making an optimal choice.

2 Formulating Solutions

Let's look through an example of a classic greedy question to demonstrate how one should approach greedy.

Known as the Activity Selection problem, given a list of events with a starting and ending time, we want to participate in as many activities as possible, ensuring that no two chosen events overlap.

The first instinct for most people is to select the activity that comes up first, then start from the end of that activity and repeat the process of finding the next closest activity. However, with a quick example we can see that this approach is not correct. This is an case where a seemingly greedy algorithm does not actually choose the most optimal solution.

Now we go for another possible solution - to pick the activity that takes up the least time first, then continuing to take the smallest activity that does not interfere. Essentially, we are sorting the length of the activities, then taking them in order if they don't overlap with previously selected activities. With another very simple example we can see that this approach is also invalid. However, we start to get a sense of what might be wrong here. This particular algorithm didn't work because we had too many overlaps. To fix this, let's try and select the activities with the least number of overlaps at each step.

After a little more thinking it becomes obvious that this approach will not work either. The questions starts to appear quite intimidating, but the solution is in fact quite simple. In order to get the optimum solution, the best thing to do at every step, the greedy approach, is to take the activity with the earliest ending time.

Why does this work when none of the other approaches worked? All of our other trials had some hidden flaw that resulted in a suboptimal overall solution, even if we thought we were making the best immediate decision. By choosing the activity that ends the earliest, we are setting up the best possible future for our algorithm.

By looking at the process for the Activity Selection problem, it is clear that good test cases and an ability to prove yourself wrong are important. It's very easy to assume that the greedy approach you have come up with is correct, but you should never stop trying to find test cases that break your approach unless you have proven your solution works.

3 Difficulty

There are two points from which greedy can be a tricky topic. The first stems from an inability to firmly declare a greedy solution, and the other comes from an inability to recognize a greedy question.

You may have heard the phrase "Proof by AC" used in competitive programming - this is a phrase used when one submits a solution without having a solid proof for its correctness, but assumes that a "All Correct" status on the grader will suffice. It goes without saying that this is a bad mindset to fall into, as you can often waste time programming the solution to a question only to find out that it does not work. It's true that proofs can be hard to formulate for some questions, but practice over enough questions and an ability to make good test cases will often be an appropriate substitution.

There are many questions that appear greedy in nature, but in reality are not. It's important to recognize when DP will work, and when it will not to avoid wasting time. A common example is the coin change problem. Given the values of N coins, what is the least amount of coins needed to create K cents? Although it looks as though we might be able to greedy by going from large to small coins, this logic quickly breaks. This question is actually solved using dynamic programming, and greedy has nothing to do with the final solution. It once again becomes evident that good test cases are important for both telling when an approach is wrong, and when a problem is not greedy to begin with.

4 Example Questions

As is the key with every programming concept, it's important to practice questions to get accustomed to the topic. Codeforces has a greedy filter on its problems that allows you to view all questions. You can specify the difficulty range, or just look for problems that you find interesting. Make sure to look at the editorial if you can't think of a good solution.

Codeforces Practice Questions