Gmail B P <beverly.pham@gmail.com>

Patterns for breaking down questions you haven't seen before.

1 message

Parker from Interview Cake <yourfriends@interviewcake.com> Tue, Mar 26, 2019 at 5:00 PM
To: beverly.pham@gmail.com

This is Day 2/7 of our 1-week coding interview email course.

Yesterday I told you about keeping an *algorithmic patterns* list as you run through practice coding interview questions.

And I promised to send over the patterns I've found to be most helpful.

Here they are:

---

1) Brute Force
"Enumerate all possible solutions, and check them for correctness."

This is usually our *first thought* when we're breaking down a new problem.

Want to get the highest product of 3 numbers in an array? Grab every set of 3 numbers ("all possible solutions"), multiply them, and keep track of the max ("check them for correctness").

This idea can be applied to *most* questions, and it's a great way to get from "I have no idea how to even get started" to "I have a way to break down this problem."

The brute force solution is *usually not the most efficient*, but it can still be a helpful way to start.

By the way, when I say "efficient" I'm talking about "big O time cost." If you're rusty on big O notation (or have no freaking idea what that is), definitely read our explainer.

2) Greedy
"Keep track of the best answer 'so far,' in one pass through the input."

This is a common way to go from brute force to something more efficient.

The tricky part is answering the question, "what values will I need to keep updated at each step in order to have the final answer when I reach the end of the input?"

Our stock price question is a good example of this in action.

3) Use a Hash Table
"Seriously, just use a hash map. Or maybe an array."

Throwing a hash map/hash/dictionary at the problem is actually the answer a *lot* of the time. If you just make it one of your first thoughts, you'll blow through these kinds of questions.

Hint: it's often a "counts" hash, mapping items to how many times they appear in the input array.

Our "top scores" question is a good example.

One caveat: this approach *spends space* in order to save time. You can impress your interviewer by noting this tradeoff! (Not sure what "spends space" means? Seriously, read that big O explainer!)

Hopefully these patterns make the algorithm design process feel less abstract. That's our real goal here: to shine a light on the black box of algorithm design.

In those moments where you're not sure how to solve a problem, inspiration doesn't just strike out of nowhere. It comes from the *patterns* you've built up in your mind.

Devoting conscious effort to building those patterns will take you a long way.

Go ahead and add these three to your running list of algorithmic patterns. You *do* have your own algorithmic patterns list that you've started, right? C'mon, grab a piece of paper and start one!

Is it starting to feel like you're getting the hang of this stuff?

There's still a lot to learn about winning interviews, but stick with Interview Cake and we'll walk you through everything until it all makes sense.

To get even more comprehensive training, buy our full coding interview prep course, where I'll break down exactly how you can use algorithmic patterns to solve any interview question.

It only costs money if it gets you the job!

Later,
Parker

P.S. That guarantee works exactly how it sounds—if you don't pass your interview or don't like the course for any reason, you can just ask and I'll give you a refund. But maybe you should see why 98% of my students don't ask for one anyways.
