csgator Follow
Oct 29, 2018 · 9 min read

Leetcode Pattern 4 | Meta Stuff

So this article is a bit different from the other ones in this series and it's not about specific algorithms but the general approach one should try to follow in their leetcode journey. This is based on my own personal experiences and what I have gathered talking to people who got multiple big-N job offers after a good amount of leetcoding. I think this article would be by far the most important one as it touches on the meta stuff that not a lot of people talk about.

Problem Solving Approach

Think about the problem for at least 30 minutes and put in your best efforts to solve the problem on your own. Think through the test cases, imagine how you would be writing code and write the code without IDE. Try to minimize the use of a compiler. It's important to write bug-free code without using a compiler for big N companies. I'll explain below why I mention these very specific points (some of which might have freaked you out).

Why the 30-minute mark?

The thinking part is really important, can't emphasize this enough! I didn't have a lot of patience early on and would just try to optimize the number of problems seen over learning. I would look up solutions after 10 minutes of sloppy efforts. After a few times of doing this, I wouldn't think at all and would just reach out for the discuss tab.

There is a bit of learning that can happen through this process but there is a huge loss too. For example, I was asked the meeting rooms problem in an interview (it is a frequently asked problem in one of the FAANG's!) and instead of thinking about the

problem at hand I would try to retrieve the solution from the buggy in-memory hashtable I built over time.

The more you struggle with a problem the more you'll learn after seeing the solution and best case is if you look at the solution after solving it on your own. Try to improve your thinking abilities, because it's very likely that when you face the same problem weeks later you'll not remember the solution and you'll most definitely think in a similar way you did the first time. So if you learned the paradigm required to solve that problem and solved a lot of other similar problems using that strategy you'll actually be able to solve it later.

The worst part is it's hard to realize if this is happening subconsciously, so as a litmus test I would say try resolving random problems from past practice after a set amount of time. Spaced repetition also helps consolidate your learnings (firecode.io is popular because of this very fact)

In real life, there is always a time constraint, interviews coming up, limited time available in the day due to coursework, and so many patterns to learn which is why it's a tradeoff and I suggested the 30-minute limit, you could play with it depending on how much time you have at hand. You need to realize that it's not about the number of problems you solved before an interview but if you are able to apply your learnings to a new problem in an interview setting.

Don't think that you need to solve all problems on leetcode that would be crazy and if you do it the correct way you won't need to solve more than 300–400 problems to be confident for a full-time interview, 150 for internships with a good balance of mediums , few hards and fewer easy (a realistic average case figure, not talking about outliers who only did 70 lc problems to land a big N offer). Most problems are repetitive anyway, same logic applied with a 1–2 line change. Focus on learning core paradigms (DP, graphs, backtracking, trees, etc)

It is fine if you're not able to solve a problem but it is important that you're able to solve a similar problem after absorbing the approach for the current problem.

Test your code

In a real interview you would tell your approach to the interviewer and if he/she is satisfied with your approach you'll be writing code, then the interviewer would likely ask you the time and space complexities and finally many interviewers ask you to write test cases. It's hard to do TDD (test driven development) and only the people who train themselves to do so can replicate that in interviews. So in my case, I would write code then write/think about test cases later.

But the best programmers I've seen would go through the test cases first to arrive at an approach. The code they write will cover the corner cases too cause they thought through the test cases first and interviewers are generally impressed with that approach. It also helps us to make sure that we understood the problem correctly and I could go on and on, just write/think through test cases first if possible :), if not at least do so before submitting your code. Avoid doing this: click submit, get WA (wrong answer) fix failing test case, repeat loop until an AC (accepted) that's a really crappy thing to do and TDD would likely help you avoid it.

Why not use an IDE?

That's totally up to you but I suggested so because many people are so IDE dependent that they are practically crippled if someday they had to write code using notepad. Google interviews take place in google docs and I understand the principles behind this weird decision. If you are really good at the language you code in and have used it to for a while, you tend to internalize the language. If you let go of the over usage of IDEs and language references you'll most likely become a better programmer but that's debatable so I'll leave it at that.

Definitely reduce compiler usage!!

Why? because when you restrict yourself to not use the compiler, you read through your code and try to mentally compile it, spotting logic and syntax errors on your own. With enough practice, you don't even make those errors and you tend to code faster. That to me is a sign of a good programmer and interviews are all about getting signals if you are good enough for the job :). This is not a strict requirement but I would say it's good to set high standards for yourself and get out of your comfort zone.

Facebook interviewers look for this skill (conclusion based on limited data points) and most interviewers are sharp enough to look at your code and tell you where your bugs

are. Can you spot and fix those bugs without compiling?

General tips

Do look at solutions posted in discuss after solving the problem or if you're stuck. You'll learn a lot by going through a few solutions, probably learn different styles of writing the same thing or learn different approaches all together. Leetcode discuss is gold and I attribute most of my learnings to discuss. Maintain a github repo where you could keep adding your solutions, an active github chart looks good to whoever visits your profile!

Don't look at problem tags until you are stuck! that's part of the problem-solving process to figure out what category the problem falls into. Alternatively to get good at a specific paradigm pick problems from that tag and keep solving until you get comfortable. It's a mix of these two opposite approaches being applied alternatively that'll set you up for success in your big N interview.

Maintain a google doc / excel sheet or use leetcode notes feature where you record why you couldn't solve it, what you learned through that problem, time to get AC including time to write code, time spent to think, number of times you clicked run code, number of submissions to get AC, etc. Such statistical analysis will definitely improve your skills over time.

Leetcode contests for the win!

I highly recommend doing contests every Saturday, because you'll be trying to code faster as it is timed and there is a ranking system based on how quickly you get AC and penalties for buggy submissions. After doing enough contests you'll rise in ranks, improve your rating and it'll give you an adrenaline rush to keep going. They also post official solutions after a few hours which are nicely written. Most leetcode contest problems were popular interview questions in the time period the contest was conducted so definitely do contests regularly :)

Lesser known fact, if you do well in leetcode contests some companies will directly contact you for an interview. If you get a rank within 200 (possible with enough practice on leetcode) that's definitely worth mentioning on your resume as well.

Applying

At one point you'll realize that you're comfortable with solving problems but if you don't get interviews you won't land your big N job. This topic deserves its own article but here's the gist:

1. Resume format doesn't matter much but the content is what matters. Keep it simple and readable, no colors or logos. Use careercup's simple template and worship this article, IT WORKS!

2. Always apply through referrals and don't think that your network is limited, use LinkedIn the right way and make connections. Most people will refer you if you're good enough cause it feels good to help a person out and also that referral bonuses are huge. It's a win-win situation for the company avoiding high recruiting agency costs, you and the person who refers you, think about it!

3. Follow point 1 and shoot your resumes to recruiters on LinkedIn or email. Many recruiters are actively looking for candidates on LinkedIn. I got my Uber interview after I contacted a recruiter directly.

4. Brands definitely matter, so once you get that big N internship/work experience it's really easy to get interviews, otherwise highlight your personal projects, course projects, open source experience, hackathon wins, whatever convinces a person in one glance that you are worth giving an interview/referral at least.

Point 4 is all about getting signals if you are good enough as a developer so whatever you can do to prove to the world that you're good enough. In my case, I didn't have any work experience so I wrote articles about solving algorithm problems and shared them on Linkedin :). I had GSoC on my resume which was an important factor and I put my personal projects on top. Keep experimenting with your resume format until it works and you get interviews.

People complain that they applied for 400+ jobs online and got no callbacks. It doesn't matter how many jobs you applied to, a more realistic statistic is to consider how many interviews you had (through whatever, cold emails, referrals, online apps). Applying online is like brute force, you can't say you brute forced enough but are still getting a TLE, come up with smarter approaches to beat the crowd.

Thanks for reading so far, you're awesome! That's it for this article and I'll end it with something that I had on my mind for a long time.

Youtube Channel

I've always wanted to make algorithm tutorials. Over time I've come to like algorithms and leetcoding is no longer an activity I pursue to get a good job. It is something I do out of my own interest, which I consider to be the best reason to do something, allows you to get really good at that thing with lower effort. Now that I have a job lined up and lighter course load I have a lot of time on my hands to actually work on this project.

More specifically I want a low effort thing where I can help people get good at leetcode style problems and understand core algorithms. I am a big fan of youtube, it is the primary activity of my day and I realize the kind of effort that goes into recording and editing videos. I plan to stick with simple screencasts and a whiteboard where I could explain an algorithm and solve a few leetcode problems using that algorithm.

Why low effort ? because I would be able to keep doing it in a sustained manner, meaning more content produced overall and more topics covered. In the past, I've received a lot of positive feedback for this series and I would love to continue this but it's a lot of effort to write articles and it's easier to convey information in the form of video. I am trying to measure interest in this project so please do leave a comment or send me a message on LinkedIn to let me know what you think about this :)

Fun facts :

1. I never had taken a formal algorithms course until this semester and my undergrad was in Electronics.
2. Leetcode's founder Winston Tang had developed the website for his own use while he was prepping for interviews.
3. Never gets old -> https://www.youtube.com/watch?v=wZR6QFE2m6o

Leetcode Algorithms Software Development Technology Coding