Contents

Resume deep dive 1
Technical interview 1 with Nikhil 2
    Templating Engine 2
    Post-mortem: 2
Technical interview 2: Question by Chin Ying 3
    Sampling a weighted list of items 3
    Allow real-valued weights 4
    Post-mortem 4
Post-technical interview 5
Wrap-up 6
Conclusion 6

Resume deep dive

I think I bombed this section. I think I chose the wrong topic to talk about, which was because I had prepared it with Joshua the night before. So everything seemed to go OK when I explained the schema and demonstrated how it could be used to represent any arbitrary analogue board state.

But they didn't seem to be impressed. First they asked "How do you encode the rules?" I said it was a deliberate engineering decision *not* to encode the rules because this allows us to represent all games. If you encode rules you can't represent games like e.g. Hearthstone. Also, it's meant to evoke the analogue feel when players enforce the rules themselves. But I don't think they were convinced—or rather, it's not as technically impressive if you don't encode the rules. I felt that they weren't impressed with the schema.

I also told them about the fact that I decided to handle promises in the "wrong" way: making the init function blocking instead before allowing any other methods to be called. The point is that we were able to isolate the Promise object into one singular method call (boardGameInit) or something. The disadvantage is an optimisation loss because it now blocks. But what we gain as a result is that the rest of our codespace can be synchronous and we don't have to worry about promise handling in the rest of our codebase which simplifies development greatly — not polluting the codebase.

"Can we see some code?" I showed them the init function. They asked me whether there was any other way to implement it, I said you could do `Promise.all().then(() => whatever)`. They asked me what's the difference between using `async` and using `Promse.all()`. I replied that it was basically making it look nicer and avoiding promise chaining. But I don't think that was a very good answer.

There was a period of silence and I could tell that they were trying to get *some* positive signal out of me. I was feeling pretty deflated at this point. I wanted to tell them about another project but I would have to tell them the context of the

project and we wouldn't have time for that. They asked for the most challenging bug that I faced and I told them about the rather esoteric bug with `os.fstat` and `os.stat` being affected by `os.seek` (which is not what the top answer on SO said—the answer was wrong). But I don't think this was that impressive either.

I think that my signal wasn't strong enough here.

Technical interview 1 with Nikhil

Templating Engine

Implement a string templating engine

1. Implement a function fillTemplate(s, m), which takes as input a string s and a key-value map m which replaces snippets surrounded by < and > with their corresponding value in m.

Do not use regular expressions to solve this problem as it makes solving subsequent parts of the problem unnecessarily difficult.

For example: s = "Hello my name is child of " m = { firstname: "Luke", lastname: "Skywalker", father: "Anakin" } fillTemplate(s,m) // "Hello my name is Luke Skywalker child of Anakin"

2. Modify your code to work with partially defined maps. If a replacement value is not given the keys in the original template should not be replaced.

For example: s = "Hello my name is child of " m = { lastname: "Skywalker", father: "Anakin" } fillTemplate(s,m) // "Hello my name is Skywalker child of Anakin"

3. Sometimes you want to actually show angle brackets instead of having them be replaced. Modify your code to allow users to "escape" characters by preceding them with a semicolon.

Before you start implementing your solution, discuss with your interviewer what edge cases there are when implementing escape characters. Then come up with a list of test cases and desired outputs based on the discussion.

For example: s = "Hello my name is child of ;<father;>" m = { firstname: "Luke", lastname: "Skywalker", father: "Anakin", } fillTemplate(s,m) // "Hello my name is Luke Skywalker child of "

Post-mortem:

This was a string replacement question I managed to get through three parts of the question.

The first part was simple string replacement: this was done pretty quickly.

The second part was not string replacing something if the key wasn't in the dictionary. This was also OK: pretty quick, 4 minutes.

The third part is where I struggled. This is about escaping the replacement with a semicolon and there are a couple of cases to think about. I got a correct solution

At the end I mentioned that the code was getting very difficult to reason about because of all the cases and checks. Nikhil agreed and asked me what I would do. I said if I had more time I would refactor it: take a good hard look at the checks and make sure they aren't redundant. I don't think that was the answer he was looking for, though. I also considered splitting up into helper functions but couldn't see a good way to do it.

Nikhil asked me what the time complexity of the solution was and I said it was O(N) or O(N^2) depending on the time complexity of replacedString += char. If the concatenation operation is O(1) then since we only walk through the string once, the overall time complexity is O(N). I Googled and checked and the first answer here seemed to say that the concatenation operation was O(N). So I wasted a bit of time trying to get str.join working. By the time I was done time was over. It turns out that there is no difference: this SO answer gives empirical evidence. So I needn't have bothered actually and should have moved to step 4...

In general, I feel like I don't do very well when the requirements of the question keep changing under me.

Before writing this post-mortem I thought I bombed this question. After writing this post-mortem I guess doing three parts in 40 minutes is not too bad. I correctly identified the time complexity and I believe my solution is asymptotically optimal. My thinking was a bit muddled when it came to the third part. And I should have thought a bit more about how to refactor the function— but I was pretty frazzled at that point.

"'

Technical interview 2: Question by Chin Ying

This is the question I was given by Chin Ying:

- 1713 started
- 1730 solved part 1
- 1750 solved part 2, concluded because of lack of time.

Sampling a weighted list of items

Select an item with probability proportional to its weight Implement the "function":

weighted_select(items, weights)

where items is a list of strings and weights is a list of integers of the same length as items. The procedure shall randomly output an item from items. The probability of each item being selected shall be proportional to its weight.

Example:

> weighted_select([“orange”, “apple”], [1, 1]) “orange” weighted_select([“orange”, “apple”], [1, 1]) “apple” weighted_select([“orange”, “apple”], [1, 1]) “apple” weighted_select([“orange”, “apple”], [1, 1]) // orange selected 50% of the time “orange” weighted_select([“orange”, “apple”], [0, 1]) // “apple” selected 100% of the time “apple”

Allow real-valued weights

Can the previous implementation be modified to support positive real-valued weights?

> weighted_select([“orange”, “apple”], [0.8, 1.2]) // orange 40% of the time, apple 60%

Post-mortem

I considered a couple of different solutions.

There is a solution that takes up in O(W) memory where W is the sum of weights but all queries henceforth run in O(1) time. There are other solutions that are more space efficient (O(I)) time where I is the number of unique elements but that wouldn't run in O(1) time.

I should have explained that I thought this solution was good because you usually sample something multiple times so it's more important that the sampling be fast rather than trying to save memory.

Had a bit of difficulty adding a test function because this function is nondeterministic. This took up some time. I suggested two solutions. One is to sanity check the dictionary to see if it's correct. The other is to do Monte Carlo sanity checking: run the function a large number of times and see if the numbers approximate the weights. Chin Ying said let's do the latter. I was able to get the idea right away but was waylaid by a (not very consequential) bug.

The extension was to support positive real-valued weights. I said that it was possible if you gave up correctness. Chin Ying asked me to explain. I said because of floating-point precision, you will never get exactly the correct weights. Whereas with integer you will always get the correct answer. He agreed.

We discussed a bit about the time-space tradeoff. He said let's assume we accept the tradeoff. I wrote a solution with the worst-case time complexity of O(I) to generate the data structure and O(I) per query, but memory complexity of O(I) as well, which is good if I << W.

The idea is as follows. Given the input `weighted_select(["orange", "apple"], [0.8, 1.2])`

we form the following sorted list:

[(0.8, 'orange'), ('2', 'apple')]

Then we generate a number between 0 and max. Call this N. We walk through the list until we find a number such that N < tuple[0] whereupon we return the number.

The asymptotically optimal solution while keeping memory complexity of O(I) would be to convert the list

[(0, 0.8, 'orange'), ('0.8, 2', 'apple')]

to a BST which gives us log(I) lookup. I would not have time to implement it, but it would have been good to mention.

I wrote this to Chin Ying:

Dear Chin Ying, Thanks for your time today. I loved your qn. I didn't mention in the interview (panicked) but the runtime could be improved with a BST.

If we had the list

[(0, 0.8, 'orange'), ('0.8, 2', 'apple')]

we can convert this into a BST which gives us log(I) lookup.

Enjoy Happy Hour! ZH

On hindsight, these questions were quite open-ended and not like the questions I practiced for on Leetcode. It was a bit sad because I feel like I could do better. Overall, though, I think I came up with an OK solution. It wasn't asymptotically optimal but the solution was quite clean and I correctly identified a time-space complexity tradeoff.

Post-technical interview

I asked questions I wanted to know the answer to:

1. What do you wish you had known about OGP before you joined?
2. What do you like about OGP and what don't you like about OGP?
3. How does OGP decide on what projects to work on?

Chin Ying and Nikhil both independently gave the same answer, which was quite funny. Both pointed to the autonomy as the best part of OGP and the worst part as the bureaucracy involved with the civil service. Nikhil also said that he feels like the team is growing a bit too fast in the sense that he doesn't get to know the new members of the team as well.

Russell answered the question how OGP decides what projects to work on. He said it's a lot about requirement gathering from government. The main difference between OGP and the rest of GovTech is not—as I previously thought—building products for the citizen vs. building products for civil service, but rather about the process: it's more of startup culture, iterating quickly, having annual monthly hackathons to iterate and ideate, rather than a more requirements-driven development process.

Wrap-up

I thought that I bombed and I couldn't bring myself to be positive at the end—I must have given a bad impression. I should have tried to stay positive and wish them a good day.

Conclusion

I would give myself a 4/10 for the resume deep dive and a 6/10 or 7/10 for both technical interviews. I would definitely not consider either technical interview "Strong Hire": only a "Weak Hire" signal. And I would not hire me based on the resume deep dive.

I don't want to be fully negative, however. I think I did an decent-to-good job asking for clarification, communicating my solution, and communicating when I got stuck. There was no need for a trace here because it didn't require much complicated algorithm. And I think I performed overall OK with the speed of the technical interview, even though I didn't finish all the parts. (There was at least a part 3 for the)

Overall, I don't fancy my chances: I think it's more likely that I don't get moved to the next round than that I do. I would calibrate my posterior as 30% that I get moved to the final round.