Hi, I'm Okwus, a technical recruiter here at Google. In this video, we’ll show two software engineers at Google working through a mock technical interview question. Please note that this example is not a question you'll be asked exactly. However, it represents the type of questions we like to ask our candidates: one that allows you to state your assumptions, improve your solution, and is ultimately open ended enough to show us how you think. If you're interested in seeing an example of solving a system design question or learning more about our hiring process, visit the videos linked in the description below. Onto the mock interview. Hello, my name is Sami. I'm a Research Scientist at Google Research, and today, I'll be playing the role of the candidate. And I'm Juliana, I'm a Software Engineer at YouTube, and I'll be playing the role of the interviewer. Just as a reminder, you won't have access to a compiler or IDE during your interview, so practice writing code with Google Docs. (Sami) Try to do it at home. Write code. Make sure your code is tested. Do this process before you come to the interview. (Juliana) And don't worry about small syntax errors like which substring to use for a given method. Just pick one, like start end or start length, and let your interviewer know. Okay, let's get started. Okay. So Sami, here's my technical question for you. A farmer wants to farm their land with the maximum area where good land is present. The land is represented as a matrix with ones and zeros, where ones mean good land and zeros mean bad land. The farmer only wants to farm in a square of good land with the maximum area. Please help the farmer to find a maximum area of the land they can farm in good land. I'll paste the question and that includes an example as well for you. (Sami) Awesome. May I take a moment to read it and then ask a few clarifications? (Juliana) Of course. (Sami) Thank you. Awesome. So my first question: must it be a square? The area they want to farm, or could it be a rectangle? (Juliana) For this problem? We only want it to be returning a square. (Sami) Awesome. Okay., thank you. (Okwus) As a reminder, it's important to get clarity whenever possible. Many questions will be deliberately open ended because we're looking to see how you engage with a problem, your primary method for solving it. Be sure to talk through your thought process and feel free to ask specific questions if you need clarification. (Sami) And— I can think of like, the naive solution, but probably I shouldn't do this, but I'll just run it by you. (Juliana) Okay (Sami) I can loop over every position. And for every position, I can basically— start from that moment onwards, go left and right, and count however many ones I can find in like the— a rectangle. or a square, sorry. But that would be if, if let's say the dimensions of the square is like n by n, that would be at best n to the four, which is not ideal. And I assume by posing this question you want me to find an efficient solution. (Juliana) Yes. (Okwus) This was a good example explaining your thoughts. Remember, we want to understand how you think so it's always good to explain your thought process and decision making throughout the interview. We’re not only evaluating your technical ability, but also how to solve your problems. Be sure to explicitly state and check your assumptions with the interviewer to ensure they are reasonable. (Juliana) I'm wondering if you can just visualize what maybe this brute force approach would look like. Maybe using the example. (Sami) Sure. Okay, so the brute force if I want to visualize it so I just pasted the example and I'm going to go over every index, [i] [j] index. And how to— the first, this is the second, this is the third, etc. Let's say we get here in this iteration. So [I] is becoming, let's say two, and [j] is becoming one. In this position, the row and column. So from this point, as every point that we go to, we start another double for loop, going to the right and bottom. So here I go one step to the right. There is a one, so I continue. And then I go to the bottom, there is a one and I continue. And then I go to the right of it, there’s a one, I continue. I wonder if that makes sense. So first we start saying, okay, when we get to this point, we have, you know, that we can do a one by one. Then we ask, can we do a two by two? And if the answer— to answer that, we have to complete the two by two triangle. So we have to look in this vicinity so to speak. (Juliana) Yes. (Sami) Then we say okay now yes, I know I can do a two by two starting from this position. Can we— can I do a three by three? And to answer that we have to go on this like, wrapper. Like I'm going to select them one by one because it’s harsh like otherwise, and I then also like those. So we— like, extend the boundary by one comma, and one row to check that they're all ones there. And then we keep track of the maximum. So if we arrived at three here, then I memorized the three is the best so far, and then until I loop over all the indices possible, maybe I'll update my three to something else and then I’ll return whatever number has been kept. But as we already said, this is not optimal, but it will do the job. (Juliana) I think that even though this may be a brute force solution, it would be the right approach in checking the nearby values and having that inform how large the square could be. (Sami) Awesome. And I guess, do you want me to program this brute force solution or am I'm going to be penalized and should I just (Juliana laughs) (Sami) think about something better? (Juliana) We can just move on to maybe what you would consider a more optimal solution. (Sami) Sure. Awesome. Okay, so these kinds of problems, I feel like I can solve them in two other different ways. The first one would be recursive. So here the recursion can say, let's ignore the zeros because as soon as I find a zero, we can just return. From this point, you can't have a square. But let's say we get to this point it’s like, “oh yes, there's a one so I can make a one by one.” Then we can ask, the surroundings, can we make more than one by one? And then, if I can ask the square next to me… Actually, let me think about it, okay. If this is a one, the right of it is a one. The bottom of it is a one. We can ask this— that I'm highlighting: how big of a square can you make it? Actually... (Juliana) No, I think you're on the right track, that you're checking the right, below, and diagonal values. But because there is a zero here, does that mean it could be a valid square? (Sami) Zero is invalid, right? (Juliana) Yes, yes. That— that would be the case. So— so based on that, what can you infer that... if that, you're at that one position at the— that index, yeah. (Sami) If I'm at the index, (Juliana) What can you infer is the maximum square? (Sami) At this index, it’s— it is a one because we can see the diagonal is already a zero. (Juliana) Yes, yes that's correct. But you then know that the neighboring values are also ones, so there is a possibility that those could have valid squares that are greater than one. (Sami) Correct. Yes. I mean, in this specific case, because the diagonal is zero, that means this next to this one below it is a zero. Unless we're coming from above. (Juliana) That’s true. I guess the one below it. Actually neither of those, but... you could potentially have, as you're saying, a recursive algorithm. How would that look then? (Sami) Mhm. So the recursive algorithm— I'm trying to think out loud as I'm describing the solution. So, basically when we see a one here, we can— we can ask can this be part of a bigger rectangle? And recursively, how would that look like? We have to ask the neighbors how much— how many ones they have consecutively, contiguously. (Juliana) Mhm. (Sami) So that could be saying that, okay, there's a one here, then we can invoke a recursive call that says how many— how many ones are starting at the right of me making a square? So if— I wish I can highlight, but let's imagine me — I'm going to bold this. I'm going to bold this. That marks the end and the beginning of a square. So when I'm at this one right here, I asked the square. I asked right of me, “How big of a square can you be?” And— if let's say this whole area had ones, even though it doesn't, if it had all ones, the right of me should say, “Oh, I can do a three by three.” And I'm going to try to break this down into maybe one step at a time. So when I ask this one, “How many ones do you have?” “How big of a square can you form?” It could ask the one right of it, “How many can you form?” It can ask the bottom of it, “How many can you form?” (Juliana) Yes. (Sami) And then, if— how— whichever— so, whichever returns the minimum, I will take that and I'll add one to it. Trying to think of this will always be correct. So if this one said, “I can do a two by two.” let's say it claims it can do a two by two. I'm going to change your input by a little bit, if that's okay. So which means this would have been one, this would have been a one. (Juliana) Okay. (Sami) And this— the one that I'm highlighting is asking the one right of it, “How big can you be?” This will say, “Oh I can do a two.” Then it asks the bottom of it, “How big of a square can you do?” In this particular case, it is going to say three. Because I can see the ones like, here. This one... (Juliana) I think that... (Sami) and this one are making a square. (Juliana) Ah, I see. So you would compute it like, top down. So that one that you've highlighted, in the— in two two index or two two coordinates would have it contain a three value because it can form that three by three square. (Sami) Right. Exactly. (Juliana) Got it. Yes, that makes sense. (Sami) And... let me see if taking the minimum is sufficient, so to speak. So this one can create a three by three, yes. The one right of it can create a two by two — or I guess now the example is so corrupted because I kept changing things. Let's see, actually, this one. Yes. So this one that— maybe I should color. So let me color this to red. The red one will say, “I can do a two by two.” Starting from this coordinate. (Juliana) Yes. (Sami) The— the blue one, or the one that will become blue, will say, “I can do a three by three starting at this coordinate.” (Juliana) Yep. (Sami) Now, how much can this one create the one I'm highlighting? It would still be the minimum. (Juliana) Yes. (Sami) I’m thinking. Right. Because you can do more than three by three. Okay. So, now— now we can think about the base cases of the recursion, so to speak. (Juliana) Yes. (Sami) So the base case— if we had a zero, then we can do— we return a zero. (Juliana) Yes. (Sami) If we had a one, then we ask the right and the bottom to do the recursion. We invoke them. (Juliana) Yeah. (Sami) And we return their minimum. (Juliana) Yes. (Sami) I'm trying to think, at which case do we— in which case do we grow? I’m thinking we can grow... Let me think. So if this— if the red one, hypothetically said, “I could do a three by three.” I'm just thinking. And the bottom one can say, “I can do a three by three.” Okay. I need to think that— there has to be a way that we can grow the one. You know, we have to return a five or something eventually. So if the right of me can say, “I can do a three by three,” which, you know, so maybe I can hack the numbers again, just for the sake of example, it can say, “I can do a three by three.” Actually, this is a still a two by two. Okay. I have to do this as a one. So the red one can now do a three by three. The blue one can also do... (Juliana) Three by three. (Sami) Only three by three. Exactly. Then this one, is so far is going to say, “Three is the number.” (Juliana) Yes. (Sami) However, if this was a one right here. (Juliana) It would be four. (Sami) Then it will be four. But let me just, like— (Juliana) So that would mean the diagonal number would also be three. To which case, because you have a one in that first field, the— at the zero one coordinates. What could that say? If all of the three values down, diagonal, and to your right are all three and this is a one, what could that be? (Sami) Makes sense. Okay, so if all the values are three, then I should say, “I can do four.” (Juliana) Exactly. Yeah. (Sami) Make sense, okay, okay. So I think— I think I'm able to write it recursively. Of course, the recursion is not the most optimal because it has to retry something multiple times. (Juliana) Yes.. (Sami) We can always add a memoization memory element to it, this way, each unique invocation will only happens only once. (Juliana) Yeah. (Sami) Which will make this be at most, quadratic. We— will invoke at most, n square times. But usually for anything that you can do with memoization, there ought to be a dynamic programing solution where you can do bottoms up. You want me to program the recursive memorization lazy version? Or should I strive for the most efficient dynamic programing version? (Juliana) I'm thinking that I want to first see your thinking for the bottom up, the dynamic programing solution, and then we can go ahead and code it. I just want to see what you're thinking would be for how bottom up would look. (Sami) Sure. Okay. You kind of gave me a hint already by saying, “check the diagonal, if it also can do a three by three.” So in that regard, maybe I can have... I can— this is the input we're highlighting right now. We can start with the dynamic programming array that has the same exact size... (Juliana) Yes. (Sami) but it's initialized to zero everywhere. (Juliana) Yep. (Sami) Then at any moment that I want to compute an entry, I want to populate an entry in this DP matrix that starts with all zeros, I'll be looking at the... the same position of the input, and also the surrounding positions of the input and potentially the surrounding positions of the dynamic programing array, the DP array, and populated incrementally as such. (Juliana) And so your DP array would contain which values exactly? (Sami) Okay. So at every... at entry in the input, if the entry in the input is zero, then we should write zero in the dynamic programing array. If the entry is one, then we need to basically look at the surrounding positions, as you said, like the top position. Just top of me. Just left of me, just diagonal left of me, in the dynamic programing array, and we should take the minimum. But if they all agree to be the same value, I'll just increment by one in the DP array. And the final return of this whole program is the maximum entry in the DP array. (Juliana) Yeah. (Sami) Does that sound correct? (Juliana) That sounds correct. I do want to highlight that you will be then capturing the height of the square instead of the area in this array. (Sami) Mhm. (Juliana) That's fine too, if that's the direction you want to go in. (Sami) Capturing the height, okay. (Juliana) Because it is a square, so... (Sami) It is a square, I see. (Juliana) Yeah. (Sami) That's right. Yes. Okay. Should I start programming the dynamic solution? And I can explain it as I program it, if that's okay. (Juliana) That sounds great. Let’s do it. (Sami) Okay, so programming Python into a document. So maybe I can choose Consolas as a font, and I can say def largest square. And it takes input as a bin\_array. I'm going to say bin\_array, is that fine? Do you want it to be typed or not typed as in like, you know this— the typing of Python? Should I do something like this: np.ndarray? Or at least say it's a list of list of int? (Juliana) Yes, it will be a list of— it will be a 2D array of ints, yeah. That's perfect. (Sami) Okay. Awesome. So... And then the answer is just an int, which is the maximum area. (Juliana) Exactly. (Sami) Okay. So first I'm going to say my DP array is equal to an error of all zeros. I guess I can— I want to capture first n, which is the length of the bin array, so it could be a five by five, ten by ten, square, etc. It's n by n. So we want to make an all zero. I'm going to do this. Okay so now after this line, DP is just an n by n array with all zeros. (Juliana) Oh, something that I wanted to point out here is that it's possible that it's m by n. The width and the height do not— (Sami) May not agree. (Juliana) Uh yes. For the input array. (Sami) Good to know. (Juliana) But your output array— or your output area should be a square. (Sami) Thank you for making this clarification. Okay, so then I will do this as my n times m. (Juliana) Perfect. (Sami) Now I can say something like this. (Juliana) Yep! (Sami) So let me just make sure. Yes, I'm consistent in my indexing so to speak, that if an index [i] [j] exists in DP, then it must also exist in bin\_array. (Juliana) Yep. (Sami) Awesome. Okay, so my DP starts with all zero. Now we're going to do the double for loop that we mentioned. So for I in range(n), (Juliana) Mhm. for J in range(m), and here at every square, we're going to write the maximum that we can do. (Juliana) Yes. (Sami) Scroll down. So. Let me think. First, if the entry had the zero, (Juliana) Yeah. (Sami) Bin\_array of [I] [j] is zero, then in the DP, we’ll have a zero in it. (Juliana) Yeah. (Sami) So in that case, I will just, because the DP was initialized at zero, I can just continue. I can skip to the next iteration. If I'm allowed to use— (Juliana) Yeah. Yeah, of course. (Sami) Awesome. Okay. So the— then at any entry that's not a zero, we should look up the left, and the top, and the diagonal, and we'll see their values. So, maybe I can say, left value is equal to right value. For now. I'll set them later. To the diagonal value all to zero. And I'll set them conditionally. Why? Because I don't want to skip the boundary conditions. So if the left of me, if [I] is greater than zero, that means I'm not at the boundary. That means the left could be something that's not zero. So now we set the left to DP [i-1][j]. Yep. And I want to do the same thing, similar thing, to the right one. Otherwise they will stay at zero. (Juliana) Quick question. Is it possible to handle the boundaries before you enter this scope? (Sami) Yes. Yes, I'm sure. So we could copy the boundaries from the input to verbatim. To the DP. And then our loop could be starting from one set from zeros. (Juliana) I'm thinking more so that if you are in the zeroth index of the row or the column, if there is a handling you could do before you enter this scope of setting the left and right diagonal indices. (Sami) So you're saying, like, before we enter these loops we should handle the borders. Maybe that's what I'm hearing. (Juliana) Maybe, yeah. I think it's possible to do it within the loop, if you'd like. (Sami) We could. (Juliana) Or let me know if that doesn't sound right. (Sami) We could do it. So you are saying that perhaps instead of having these if statements inside the for loop, I could do something before then, I no longer need these if statements. (Juliana) You could do it outside of the for loop, you could do within the for loop. Either way. (Sami) I see. Okay, so yes, I could do that. I mean there are many ways to do it. So, I want to think that, yes, okay. So how about, I wonder if it will complain? Okay, so mathematically speaking, I can... do this left— Let me know what you think about this. So what if I set it to that and then I just multiply by a Boolean expression? [I] is greater than zero. So here, what I'm trying to do is to say, the I greater than zero, this boolean condition, will only be valid, will only say one, if I'm not at the boundary. If I'm at the boundary, then I can say your left a zero. Is that okay— (Juliana) Yeah, that's possible. (Sami) That’s possible. (Juliana) But then if the value is one? Then the maximum square could still be one, even in the row and column, like the edges. (Sami) Right. So I still— so yes, here, like this right and left and diag, so far, they're just for reading. I still haven't inserted anything into the DP. (Juliana) Okay. (Sami) My plan is to insert in a moment, like, perhaps right now. Awesome. (Juliana) Okay. (Sami) Okay, so if [I] is greater than zero and [j] is greater than zero, and now I can read my diagonal. Diag is equal to— I'm going to copy the line for, I’ll just add a minus one. And now I want to write the DP. (Juliana) Okay. (Sami) At this moment right here. So we, just like, we’re brainstorming if they're all— if they’re all set, then I'll read the minimum of the three and increment by one. If they're not all set, then I'll just read the minimum. (Juliana) Yes. And what would be the deciding factor for that? Whether the... Yeah, what would be the deciding factor of that? (Sami) So to increment by one? (Juliana) Yes. (Sami) Basically, the... the array entry should have a non-zero. (Juliana) Got it. (Sami) We’re anchored by one. And that's the first if statement in the nested for loop that if we are at zero then we just skip that entry, we keep it in the DP as zero. (Juliana) Okay. (Sami) So if the code flows until this point, then here I know that we— it's at a non-zero point, and the top, the left, and the diagonal have some numbers. And using those numbers, the top left or right, some of the could be zeros. The top left diagonal. (Juliana) Yeah. (Sami) Some of them could be zeros. If they are all set, that means I should take the minimum plus one. (Juliana) Okay. (Sami) Makes sense. Right? (Juliana) That makes sense. Yep. (Sami) Awesome. If left is greater than zero, and right is greater than zero, and diag is greater than zero, then I will set the entry right here. DP[i][j] to the minimum of those. Left right, I think I should put in an array, syntax wise? I forgot, but I can try that. Plus one. Okay. Let me think. (Juliana) Is there? Wait. Sorry, quick question. Is there a reason why we need to check that all the values are greater than zero if we are taking the minimum here? If we've already checked that it's not a zero value, do we need to actually check that? (Sami) We don't need to. You are correct. Yes. So this if statement is irrelevant because if any of them is zero, then the minimum will also be a zero and the final answer will be a one, which is exactly what we need. You’re right, this segment is not needed. If anything, it might make a bug. (Juliana) No, that's perfect. Yep. (Sami) Awesome. So we still don't have the final answer. The final answer is keeping track of what was the maximum. Of course, I can stay within the same computational complexity, and I can just say, return the max over all of this. (Juliana) Yep. (Sami) So return, I guess I'm going to say rowwise\_max is equal to max(row) for row in DP. And then I can return the max of rowwise\_max. So row wise max in this line, basically it becomes an n numbers. Each number is the maximum in that row, and they return the maximum of those. So we have the ma— we're invoking the max function once in this for loop, and once again on the— list. The final list. (Juliana) Is there a way to potentially get the maximum or track the maximum within your for loop? (Sami) Yes, there is a way. I can do that, awesome. Thank you. So I can start as the max\_num as negative one. And here, every time I update, I basically read off the max. So I can say max\_num is equal to the max of the previous max\_num, comma DP[i][j] and then return that at the end. And of course, if there is some, you know, with every code there is ways to optimize it. Like for example, specifically here we are storing in an index of an array. We're reading again the same index. I could have stored this as a variable z or temporary variable. But I'm hoping the compiler will take care of that for me. Will make it optimal. (Juliana) Definitely. Yeah, this solution looks good to me. (Sami) Awesome. And I like the fact you kept correcting me and gearing me towards the better solution. You are a good interviewer. It was you. You were always on the right track. (Sami) Thank you. (Okwus) I hope this gives you a better understanding of what it's like to answer technical interview questions at Google. As a reminder, be sure to explain your thoughts, get clarity when possible, think through how to improve your answers, and practice ahead of time. Thanks for watching and good luck on your interview.