Facebook Interviews Technical Interview Questions +2

What graph topics should I study in order to be adequately prepared for a Google software engineer interview? Would it be worthwhile to also study algorithms for minimum spanning trees, maximum network flows, bipartite matching, etc.?

http://steve-yegge.blogspot.ca/2008/03/get-that-job-at-google.html

This question previously had details. They are now in a comment.

Answer Request ▾ Follow 174 Comment 1 Downvote

5 Answers

Anton Dimitrov, Founder & Interviewer at HiredInTech.com
Answered Mar 28, 2014

As other people mentioned the two popular graph traversals are very important just because they have so many useful applications in real-life work. However, it's not just about knowing the algorithms.

For graphs, it's super important to be familiar with the possible representations. Some that come to mind are:

1) A matrix where rows and columns correspond to nodes and in cell (i,j) there is a value representing if there is an edge between nodes i and j, and potentially showing the weight of this edge
2) List of neighbors for each node
3) List of edges in the graph
4) I challenge you to think of some more and write as a comment below :)

Depending on the problem, each of these representations could be useful. Now, try to think of how DFS and BFS would be implemented if you had to use any of these three representations. Think about the time and memory complexities of the algorithms in each separate case. There will be some significant differences that need to be taken into account.

In summary, don't just think of these traversals as one-version algorithms.

There is more to it and these are things that need to be known for your interviews. This type of knowledge is not something that needs to be learned by heart from the books. It's more about the thinking of the interviewee and how well they can choose the underlying data structures based on the problem in hand.

Now, what about other algorithms? My personal opinion is that one needs to have an idea about how shortest paths are found in graphs. After all, BFS is one algorithm that can be used for that, if there are no weights on the edges. Hell, it can even be

Related Questions

When can you say that one had prepared well or adequately for an interview with Google for a software engineer role?

Do I need to study mathematical proofs in algorithms for a software engineer interview with Facebook/Google?

I have my software engineer intern interview at Google in less than a month. What should be my strategy for preparation and how should I prepa...

What kind of courses or topics of study would prepare a programmer for the programming puzzles that companies like Facebook and Google use in ...

Is there anyone preparing for Google's software engineer interview?

What topics should I review before my Google interviews (software engineer new graduate)?

What are some of the most important topics to study for a Google software engineering internship interview?

What should I expect in a software engineer intern interview at Google and how should I prepare?

How should I prepare for my Google host interview for a software engineering internship?

What's the best way to prepare for an engineering interview (non software) at Google?

+ Ask New Question

More Related Questions

Question Stats

173 Public Followers
114,969 Views
Last Asked Jan 20
Edits

have mentioned, every now and then these things are asked. Probably not that often. I would definitely go ahead and make sure I understand well how they work. These are some very simple algorithms. I would strongly suggest you make sure you implement them on your own. This will give you some huge confidence in your skills.

As for the part of your questions "Would it be worthwhile to also study algorithms for minimum spanning trees, maximum network flows, bipartite matching, coloring algorithms, topological sorting, etc...?"

I don't think the chances of getting these at Google are high. In fact, I think chances are very very low. The only exception in this list is topological sorting. It is the type of algorithm that you don't need to know from the books to come up with. It happens to be useful more often and is not hard to learn and implement.

For the others, if you have the time, I would suggest you familiarize yourself with what they do, so that you could eventually have a conversation about them. But having to implement them under the strict time constraints of the interview seems quiet unlikely to me.

Finally, not sure if you include trees in your question about graphs, but they are definitely a topic, which will come up at interviews. There are a few more things about them that a software engineer needs to know before their tech interviews.

Jimmy Saade, software engineer at Facebook
Updated Jul 30, 2015

I actually disagree with the other answers (including Gayle's - no offense, of course) due to my own experiences, and I really hope people looking for an answer to this question read mine so as not to get the wrong idea from the other answers and go into the interview unprepared. Skip to the last paragraph for a TL;DR version of the answer.

You should know **way more** than just Breadth First Search and Depth First Search, in my opinion.

You should know, in decreasing order of importance:

1. BFS
2. DFS
3. Topological Sort & Shortest-path in a DAG
4. Dijkstra's algorithm
5. Bellman-Ford

With the exception of Floyd-Warshall, *I have seen **every single one** of these topics come up in a Google interview.*

BFS & DFS are self-explanatory. There are a huge number of variants that can be asked using them.

The rest are more subtle.

There are many problems that can be reduced to the topological sort or shortest-path in a DAG problem (heck, I've even seen questions that require you to find the longest path in a DAG), and I personally find these problems awesome, because they're seriously not obvious, but if you figure out the trick, you've solved it - if and only if you know the algorithm for shortest path in a DAG or topological sort beforehand. Now obviously you could figure out the algorithm in the interview as well (it's not that hard, but I would say it's not easy to figure it out in an interview), but why make it hard on yourself? Personally, I'd expect any SWE candidate to have taken an algorithms course (i.e., read the important parts of CLRS), and any algorithms course covers topics 1-5 above. And I'm not even an interviewer - imagine what interviewers expect.

Consider this question:

> You're a photographer for a soccer meet. You will be taking pictures of pairs of opposing teams. All teams have the same number of players. A team photo consists of a front row of players and a back row of players. A player in the back row must be taller than the player in front of him. All players in a single row are on the same
>
> 1. First, design an algorithm that takes as input two teams and the heights of players in the teams and checks if it is possible to place players to take the photo subject to the placement constraint. Then, generalize your solution: if you are given n teams, how can you determine the largest number of teams that can be photographed simultaneously subject to the same constraints?

Source: Elements of Programming Interviews - excellent book, by the way - covers everything that might come up (except possibly A*), including all of 1-5.
(I'm not going to give away the solution here, message me if you want it and can't figure it out.)

I'd probably consider asking this question if I was interviewing. It has two parts, one which is considerably simpler than the next, so it can rule out the weaker candidates. The next part (the generalization) is more challenging, and tells me that a candidate can abstract a problem well and then work with the abstraction to solve the problem. Add in someone who is able to code this solution, and you've got a good candidate. Now, I've never interviewed anyone, so I'm just talking here and many actual interviewers might disagree with me, but I think this is all reasonable.

So, now I've shown that topological sort and stuff are possible. (As an extra note, the above problem is one of the easier ones I've seen that deal with DAGs. I've seen a much more difficult one, where it's very non-obvious that you can change the problem to a shortest-path-in-a-DAG problem, but that's the exact solution. And it was asked in a Google interview. I can't disclose the question, though.)

So what about Dijkstra and Bellman-Ford? Surely they're more specialized, so it's unreasonable to ask them as many might not know them?

Ummm.... no. I barely just graduated from university, and I expect anyone who knows

And, in fact, I've seen two questions where the solution was Dijkstra's algorithm, asked in a Google interview (and the candidate had to code it). (Obviously they weren't "find the shortest path in a weighted graph". They were variants, but the bulk of the solution was Dijkstra with a small modification. If you know Dijkstra, the solution wouldn't be hard for you. If you don't know Dijkstra, you're gonna have a hard time.) And better yet, they were followed up with a question about negative weight edges, so you need to know that Dijsktra won't apply then, and you'd need to switch to Bellman-Ford - so you need to know Bellman-Ford.

On top of that, one question I've also seen asked in a Google interview was to find the shortest path in an unweighted grid (where there are obstacles or something - I forget if there's something more to the problem). The natural solution is BFS, right? Right, but after that the interviewer asked for a faster solution. That's right. Faster than the linear-time-BFS. The answer? A* (look it up and you'll get it), which is very similar to Dijkstra.

I've never seen a question that required Floyd-Warshall for a quick solution, but you might as well throw it in there since it's just one more and, based on the above, you really don't know what you're gonna get, especially at Google, who seem to have a huge number of seemingly random questions at times.

I'd be seriously surprised if anything other than these topics were asked on graphs (i.e. matching/max-flow/coloring). Why? You may ask. I just said the interviews are very unpredictable. The reason is that they're much more specialized than shortest-path algorithms, and most undergraduate Algorithms courses don't cover them, so

lanni specialized knowledge, it's unreasonable to assume you know them. Unlike Dijkstra and the others. It's also much harder to code them in 35 minutes (let alone variants of them), even if you remember the max-flow algorithm.

TL;DR: *You don't know what will and won't come up in an interview.* My answer may have selection bias, but the bottom line is that empirically, whether or not these questions are "good questions" (I'd argue that they are, but that's another debate), these questions *are* asked. How frequently, I cannot say, but I'd guess that since I've seen so many graph algorithms asked in such a small subset of people, they're probably frequent enough that you can't ignore them (and yes there's selection bias here, but if you, like me, like to be well-prepared, the only thing that will matter is that they *can* come up.) So study all the above topics. They're not that hard anyway, and any software engineer (or anyone who has taken an algorithms course, really) should be familiar with them. You can forget them after the interview if you don't need them later on in your life, but for now, you better study up. Why take a risk?

Gayle Laakmann McDowell, Ex-Googler, Author of Cracking the Coding Interview
So, here's the thing: you can learn those algorithms. Why not, right?...

Gayle Laakmann McDowell, Ex-Googler, Author of Cracking the Coding Interview

I have never seen anything about Dijkstra's algorithm asked. It'd be a pretty stupid question.

First, very few interviewers would know it. It's not like they go home every night and re-read CLRS or whatever their algorithm book is.

Second, it'd be way too hard. For candidates who were interviewing with me and finished a difficult question with some time to spare (that is, doing very well), I would ask them how to delete a node from a binary search tree. Only about 20% of candidates remembered this -- and this is of the really good ones. You think they know Dijkstra's algorithm?

Third, it wouldn't show anything. Knowing an algorithm says that you have a good memory or that you studied some stuff. We don't care about that stuff. We care about your ability to solve *new* problems.

The same goes for all these other fancy algorithm problems. Seriously, this is not what tech interviews are all about. The actual knowledge required is really just basic CS knowledge.

But here's what you should do: ignore my advice. Go look at actual google interviews on CareerCup and Glassdoor. Go make your own decision about how important it is to know fancy algorithms. (Do not look at blogs on interview questions. Best case, they have a huge selection bias and are intentionally picking the ridiculous questions. Worst case, and more likely, they're lying or pulling their info from someone who lied.)

There's a reason I don't cover advanced graph algorithms in Cracking the Coding Interview: this stuff doesn't make it into interviews.

The idea that Google asks such things largely comes from an incorrect assumption that a highly selective company must require advanced knowledge. No, not exactly. They just set higher standards about your ability to solve new problems with basic CS knowledge.

94.6k Views · 748 Upvotes · Answer requested by Ibrahim Sheikh and Arvind Shrihari

Upvote 748 Downvote

Jimmy Saade
Hi Gayle, I wanted to suggest an edit for this answer, because I feel many candidates ...

Top Stories from Your Feed