A Course to Prepare Computer Science Students for Employment Interviews

A Technical Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Harun Feraidon

May 13, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Harun Feraidon

Technical advisor: Farzad Hassanzadeh, Department of Computer Science
Jack Davidson, Department of Computer Science

A Course to Prepare Computer Science Students for Employment Interviews

- Harun Feraidon -
- Computer Science -
- University of Virginia -
- Charlottesville, VA, USA -
- hf6mw@virginia.edu -

ABSTRACT

Many computer science students often go through college courses with the intention of growing skills to get them into industry jobs. However, students often have to prepare for their interviews themselves, and they find that they are unprepared for it, despite taking all the necessary courses already. My goal is to design a course to teach students how to translate their existing knowledge into interviewing skills. It is widely believed that students who have done the most consistent practice are the best prepared. The question is which practice methods are the most important. My inspiration for this course is because it is something that I would have enrolled in early on, and it would have made me more confident in interviewing abilities. I think this will be very beneficial to other students also. I intend to research what kind of skills are most useful for interviews, and how to apply them. My findings will help people prepare for their interviews and increase their competitiveness in candidate pools.

There will be detailed information about the structure of the course in further sections.

Introduction

The motivation for a course in interviewing is that entry level software engineering roles are known to be focused on a particular type of skill. This skill revolves around being able to apply knowledge of data structures, algorithms, and occasionally system design. Interviews often come in the form of technical tests, phone screenings, and whiteboard interviews. The questions are written to not be so intense to take much time, as the interviews are usually about an hour long. Interviews often give basic information, assumptions, and a simple goal to the candidate. That being said, they are not items that a computer science degree will focus on preparing you for. A computer science degree focuses on preparing you for the actual job, but it would be helpful to students if they had some professional opportunity to prepare for the interview. Similar to how high school students often take extra-curricular programs to train in college entrance exams, such as the ACT and SAT exams.

Currently, this problem is solved by the student itself. It is up to the students' initiative to use non-academic material to figure out how the interviews work, and how to be successful in an interview. Information on how interviews work can be gained from older students or alumni, who have already faced the challenges of interviewing already. Or, online resources that the student will find that gives them the information they need. The capstone project aims project is to help all students be prepared for software engineering interviews, and not only the students who have been warned to prepare or have taken the initiative to look ahead.

It is difficult to give a concrete answer on how long it may take for a student to independently reach a point where they feel prepared for technical interviews. Often, they may feel prepared but are not actually. Vice versa may also be the case. No assessment is one of the faults of independent learning, whereas an academic course that is well prepared and organized can give the students the feeling of being prepared, should they be content with the course.

Another important thing to note is that a strong aspect of being prepared for technical interviews is taking an actual mock interview. Doing so can be difficult independently, as it is not an accurate simulation of an interview since an interview requires more than one participated person. University of Virginia's career center offers mock interviews often, however, these are done on the office's schedule, and not the students [1]. In an academic course, students can meet other students and will be able to seek the opportunity to set up mock interviews themselves.

Background

This course is designed in the structure of a student-led course at UVA, as it is also intended to hopefully be taught by a student. It is designed this way for several reasons. Firstly, there likely exists a UVA CS student who has recently taken the initiative to prepare for software engineering interviews. Furthermore, they likely may have success in interviews as proof that they are capable of. Likewise, this student may find that teaching a course like this would be a good way to retain their knowledge and remain in practice. These student-teachers would include those who have received job or internship offers from various tech companies that the students can respect. If a UVA staff were to teach this course, they may find that although they have strong knowledge of data structures, algorithms, and system design, they may not be practiced enough to be able to work through interview-style questions. In summary, an optimal student-teacher would be one who has recently independently prepared for the interview process and feels comfortable with their knowledge, they may have internship/job offers that the course's students can respect in a

teacher who is teaching how to interview, and finally, if they were a teaching assistant in the past, that would be the experience necessary to sympathize with their students.

Related Work

The most relevant related work is the popular existing method in learning how to prepare for technical interviews. This is by reading through the textbook, Cracking the Coding Interview, which will be an optional supporting textbook for this course. The book is relevant because the book is popular among computer science students for being a good resource to provide insights on how the interviews work and what the interviews may ask. I used this textbook as an inspiration on how to best order the topics learned week-by-week [4].

Another relevant work is "Mastering the Software Engineering Interview" by Coursera. This is a 21-hour course which places a strong emphasis on how interviewing skills itself, rather than training the data structures and algorithms that are usually asked about in the interview. The course includes topics such as whiteboarding through a solution, communicating your answer to the interviewer, maintaining eye-contact and speech to use, and other related skills. This course is also designed to be for computer science students. [3]

The large difference with these two related works and my course is that though they are good resources, the students themselves have to take initiative to complete these courses. They have no "teachers", only themselves. The lectures and assignments are generalized to fit the average student, which although is a good design, can be flawed when students need extra attention, such as asking clarifying questions or asking for further content.

Another disadvantage of being independently taken is they eliminate the classroom setting. Practicing for coding interviews can often be a better experience when they have classmates to do it with. Not only do these classmates have similar goals, but they can help each other study and succeed.

Class Design

Syllabus

Approaching building this course, I knew I should build a syllabus to outline plans for the course. I will use this section to explain parts of the syllabus and the decision process in them.

Rational + Goals section: The syllabus starts with two sections titled rationale and goals. The rationale section summarizes that the inspiration for this course comes to provide students a professional setting to prepare for interviews. The goals section outlines what the goals of a student in this course should be, so that students know they are in the class for the right reason.

Format section: This section states that the course will meet once a week, for 1 hour and 45 minutes. I made this design decision because this is what student-taught classes have historically been at UVA. I wanted to match that criteria to prevent any future syllabus changes if times met per week would change. This section also outlines how lectures will differ. Each week, the course will advance to the next interview topic. This approach ensures enough topics are hit, so it can feel fast paced. Some topics are rather large, such as trees + graphs, and are planned to span more than one lecture. Each topic will involve a data structure and any algorithms that should be taught with it. There will also be a week dedicated to system design, but more on that later. The format also specifics that with each topic taught, the lectures will go over interview questions after explaining the concepts, as a practice problem.

Assumptions: Here I emphasized that this course will be taught with the assumption that students feel comfortable with topics from CS 2150 (Program and Data Representation), CS 4102 (Algorithms), and CS 3240 (Advanced Software Development). I decided not to make them strict pre-requisites so that class does not become exclusive. If the class becomes popular, it may be necessary to make them pre-requisites to limit enrollment. Also, with each topic, there will be a brief review during the lecture, which will be repeating teachings from these courses.

Grading section: The grading distribution was 40 percent attendance, 50 percent assignments, and 10 percent "final interview" assignment. The "final interview" assignment will be a casual replacement to a final exam. The class puts heavy emphasis on attendance because attendance can be counted as "practice", which is important for this class's teachings. For the same reason, assignments will be worth 50%. It is only worth 10% more because I think they are more important practice than attending the lectures.

Homework Structure: I chose to assign a weekly assignment that focuses on the lecture topic of the week. This is to keep the students involved in the course and maintain consistent practice. Consistency and repetition will be important for students' success in the course because interviews will be a test of how natural is critical thinking to you. Just to encourage collaboration and meeting interviewing partners, students will be allowed one partner to work on per assignment. This decision came because the assignments will be most helpful when done independently, so adding a third member makes it risky that there will be an uneven distribution of work, and thus uneven learning. I will elaborate more on homework design later.

Final Interview (exam): Instead of a traditional final interview, students will be paired with a classmate to conduct an interview. One student will be the other's interviewer, and they will switch roles so they both experience the interview. The decisions for the final interview are difficult to set immediately, it will highly depend on how successful the class has been so far. However, the base structure to build off of will be that the final interview will be created by the student, either writing their own questions or finding questions from outside resources. These resources can be online sites that focus on tech interview questions, or the course textbook which provides many examples of interview questions. The student will conduct the interview and "grade" the student on performance metric. The interviewees goal is to impress the interviewer enough to receive a job offer. The criteria is also not yet decided, but the base to build off of will be these questions:

1.) Was the (the candidate) able to find a minimal solution?
2.) Did they find an optimal solution?

3.) Did they consider runtime or space efficiency and were they correct?
4.) Did they communicate through the problem effectively?
5.) Did they ask for hints? And if so, were they able to progress with the hints?
6.) Would you work with this candidate?
7.) Would you recommend hiring this candidate?

This set of questions will be yes or no questions. Something to consider here is some questions can be quantified, such as question 7, the recommendation to hire the candidate. A minimum value would mean strongly no, a middle value can be maybe, and a maximum value would represent strongly recommend. The answer to these questions, although important, will only be graded on completion and not on yes/no answer.

Students would grade both themselves and each other on their performance as an interviewer and an interviewee. This can be problematic as the students may find it difficult to grade their partner harshly. This is okay because the class is not designed to stress students in the grade. To solve this, there will be a required question where the interviewer gives their candidate honest feedback and recommendations for improvement. The only other question will be a review of one's own performance as an interviewer and interviewee. This section will also be graded on completion.

So how is this final assignment graded at the end of the day? 50% of the grade will be the yes/no questions completion, and the other 50% will be the feedback and self-review questions competition. So in total, all students should easily get 100% on this assignment as long as they attempt to complete it.

Also, some questions can be added or removed. This will all depend on how well the students are reflecting on the course, so it cannot be decided this early on (course design stage).

Textbook, Honor Code, and Accommodations: As per usual, the syllabus also outlines these three items. The optional textbook will be the popular "Cracking the Coding Interview" text by Gayle McDowell. It will likely be a helpful aid, but there is no intention to require students to use it in this course.

Course Schedule: This is perhaps the most important part of the syllabus. Here I outlined the plans for topics to learn for every week in the course. The schedule is of course tentative and may change based on students' needs and desires, but this is unlikely.

The first two weeks will start light. Week 1 will be an introduction to the course, where after reviewing the syllabus, the course will define what tech interviews are. Essentially explain what the process looks like: phone screens, whiteboarding, coding vs behavioral, etc.) The second week will introduce tactics the interviewer should use when in an interview. I think it is important that this is defined before the tech concepts begin so that they can implement these tactics throughout the semester.

Weeks 3 and 4 will begin the computer science topics phase of the course. Because it is the first unit, it will start with the simpler data structures, arrays and strings. In week 3, it will be mostly focused on strings, but also involve the basic array and 2D array. Week 4 will ramp up with the more complex topics: linked lists, hash tables, and the two-pointer strategy and binary search algorithm. The plan is to define the two algorithms and teach how they can be applied to strings and arrays. There will be example of problems at the end of class for the class to solve together.

Week 5 will also be an easy week, as it will show students that math-focused interview questions are also common. The problems here will be mostly easy, with a concentration on the use of modulus will be shown, a common interview solution.

Week 6 will ramp up to introduce stacks and queues to students. Students will be taught how to use them and when to use them, particularly when last-in-first-out and first-in-first-out ideas can be seen in the interview problem.

Week 7 and 8 will slow things down and focus on system design-related questions. This week will essentially focus on training students for the not-as-technical type of problems. The relevant topics I selected for these two weeks are agile/scrum methodology, testing, object -oriented programming, and threading. Agile/Scrum methodology and testing are both covered in CS 3240. The other two topics are covered in intro CS courses at UVA. I chose these upon the research of what kind of non-coding questions can be common in interviews. These topics were the results [4].

Week 9 and 10 are focused on linked lists. Although linked lists are not a complex data structure, they are commonly used in interviews and have multiple common algorithms that they can be paired with. That is why this topic is spanning two weeks instead of one. Specifically, I want to cover the "runner" algorithm, which is traversing through a linked list at varying speeds. As well as strategies for deleting an element in a linked list, which can come up in various interview questions, and is not a trivial task.

The course's learnings will end in weeks 11 and 12 with trees and graphs. These two data structure types are rather large, and there will be multiple searching algorithms taught along with it.

Finally, in the course's last week, students will take time to complete their final assignment, the "mock interview". There will likely be no required class this week to give students time to conduct their mock interviews. Or students can come to class and use that time to complete it.

Sample Homework

At the end of every unit, there will be an assignment testing students' grasp of the material.

I intend for these assignments to be both a means of practice, as well as a spot-check of material. This means there will always be at least these two types of questions: a question checking that students properly grasped the definition of an important concept and a question that sets up a coding problem, where the answer must include an implementation.

In this sample homework, I started with the "spot-check" sort of problems.

PROBLEM 1 *Explain the following, 5 points*

a.) Scanning an entire 1D array (visit every element once) will have what time complexity? 1 point
O(n)

b.) Scanning an entire 2D array will have what time complexity? Assume array of dimension arr[n][m], 1 point
$O(n * m)$

c.) When reading and understanding a problem, what can be an indication that you should the following data structures, instead of arrays:, 1 point each
i.) Lists: When wanting to append/remove elements without reconstructing array
ii.) Hash Tables: When avoiding duplicates, when wanting O(1) search + insert

d.) When might it be useful to sort an array. How fast is .sort() on Java arrays and lists? 1 point
You may need to sort an array if the problem gives you an unsorted array, and you want it to be in increasing or decreasing order for your convenience. $O(n * logn)$

From the lectures, students will learn about the importance of considering run-time complexities. The first two questions will start by testing their knowledge of run-time for traversing an array. Through this question, they remember what an array is and reinforce in themselves about how to traverse through one. Next are questions about lists and hash tables. Specifically, they are tested on why these data structures are beneficial over traditional arrays. The last question asks them to consider sorting an array, which can often come up in an interview question. It is important to remember that common languages have a sort() function already built-in, and they need to know what the run-time complexity of this would be, so the interviewer will know they considered it throughout their solution.

The next two questions require some critical thinking because they are coding questions. The first one is intentionally easier than the second one, so they can not feel discouraged by starting with a harder problem. The intention is to always include one easier problem and one rather challenging, but still doable problem.

Because the two problems are rather complex, the grading for it is broken into parts.

2 points - BF explanation
1 point - BF implementation
1 point - mention of BF time/space complexity
3 points - better solution explanation
2 points - better solution implementation
1 point - mention of better time/space complexity

There will exist a brute force solution and a better solution. Students will be taught in the second lecture to always consider the different levels of solutions in coding problems, and this idea will be drilled into them throughout the semester.

The brute force solution will always be worth less than the more elegant solution, for every problem that there can be found more than a brute force solution. This is to encourage students to seek stronger solutions. If stronger solutions were not worth more, then it would be easy for students to decide that the difficulty of the problem is not worth the points gained in the problem.

For each coding problem, there will also be an explanation of the solution, implementation of the explanation, and explanation of runtime/space complexities. Each of these will also be worth varying points. The implementation will always be worth the most because it is what the interviewers are looking for. The explanation would be worth the second most amount of points, as they supplement their code and thinking. Finally, the runtime/space consideration will be worth the least points, but still worth points nonetheless. This is so students understand the most important parts of an interviewer's question.

Conclusions

Throughout working on the syllabus and the assignment, I learned a few points about how to make this course successful.

1.) Do not discourage students by making it too difficult. Technical interviews are challenging at first because it requires on the spot, critical thinking. The course should start easy and gradually get more challenging. That is why the course should start with arrays, instead of graphs (different levels of difficulty). It is also why the homework assignments should have some easier questions that hopefully, all students will earn points on. However, there should at least two coding problems on each assignment. One with an easier difficulty and the other being more challenging.

2.) Introduce the data structures, and then the algorithms. Introducing the algorithms immediately could leave some students clueless because they lack the background knowledge.

3.) Require attendance, missing a lecture for this course would result in missing a whole unit. It is why attendance is also worth so many of the final grade.

4.) When grading assignments, make sure to grade using the guidelines. Otherwise. there may be unfair grading.

There could perhaps be more to learn once the course would be taught, and the professors would adjust along the way.

After showing this course layout to several classmates, the overall impression is that it is well designed and they would strongly consider taking this course if they were focused on training interviewing.

There were a few ideas I got from explaining the course to students. One of them is the "mock interview" assignment that replaces a final exam. Originally, it did not make sense to create a final exam, as interviews involve multiple people, and asking them to interview questions during a final exam would essentially make it a CS 4102 Algorithm's exam, which can be problematic and too difficult for this course. Replacing it with an interview is perfect because it can be low-stress and shows the student how much they've learned throughout the course.

Future Work

There is further work needed to be done for this course. For example, other homework assignments need to be designed. The professor would have to create slides to go with the material.

Beyond an actual course, I can see this project being distributed outside of UVa's network by publishing the material online. The lectures can be recorded and uploaded onto YouTube, and a Github repo can be created to publish the slides and the homeworks. The only flaw in this idea is that it would remove the incentive of taking

this course in a professional setting, and it would thus become an independent study process for future students.

REFERENCES

[1] Before the Interview. (n.d.). Retrieved November 27, 2020, from https://career.virginia.edu/interviews/before
[2] Container With Most Water. (n.d.). Retrieved from https://leetcode.com/problems/container-with-most-water/
[3] Minnes, Mia. "Mastering the Software Engineering Interview." Coursera, UC San Diego, www.coursera.org/learn/cs-tech-interview.
[4] McDowell, Gayle Laakmann, 1982-. Cracking The Coding Interview : 150 Programming Questions and Solutions. Palo Alto, CA :CareerCup, LLC, 2011.
[5] Two Sum. (n.d.). Retrieved from https://leetcode.com/problems/two-sum/