Advanced Algorithms for Undergraduate Computer Science Students

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Edward Lue
Spring, 2024

Rosanne Vrugtman, Department of Computer Science

Advanced Algorithms for Undergraduate Computer Science Students

CS4991 Capstone Report, 2024

Edward Lue
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
eyl4qaq@virginia.edu

ABSTRACT

Companies' expectations of students' algorithmic knowledge grows every year. To keep up with this demand, I propose an Advanced Algorithms course that expands upon the currently existing Algorithms class taught at UVA, which is now called Data Structures and Algorithms 2 (DSA2). The class will focus on 3 topics that are important in coding interviews and are currently not taught in the Algorithms course: range query data structures, randomized techniques, and string algorithms. The success of the class can be evaluated by examining the performance of students in algorithmic coding interviews. We would expect that students in the class would be more likely to pass an interview. After the class is created, the class must be kept up-to-date. The algorithms taught in the class should be updated with the evolving needs of the students and demands of employers.

1. INTRODUCTION

For many UVA Computer Science students, an important goal for their undergraduate education is to prepare for job interviews. Most software engineering jobs will have several algorithmic interviews throughout the interview process. It is common for students to prepare heavily for these interviews as they are generally considered the most challenging rounds in the interview process. To assist in the study process, websites that provide practice questions such as Leetcode have exploded in popularity.

At UVA, one of the required classes for undergraduate students is Algorithms. Ideally, this class provides the necessary algorithmic knowledge for algorithmic interviews. However, over time, UVA has fallen relative to other universities in coding ability. We can see this in UVA's performance in international competitions such as the International Collegiate Programming Contest (ICPC). From 2009 to 2016, UVA qualified for the world finals, the highest level of the competition, in six of the eight years. However, since 2016, UVA is yet to have another appearance on the world finals stage (cphof.org, n.d.). In addition, in the most recent CodeSignal report, UVA does not rank in the top 50 US universities for algorithmic coding ability (CodeSignal, 2023). To keep up with the competition, UVA needs a higher level algorithms class.

2. RELATED WORKS

Several universities have higher level algorithms classes to draw inspiration from. The MIT "Design and Analysis of Algorithms" class (MIT OpenCourseWare, 2015), which is their second level Algorithms course, contains a large three-lecture unit on randomization. Besides this, the primary units for the class are advanced dynamic programming, divide and conquer, incremental improvement algorithms such as max flow and matching. There also are several lectures at the end of the class on non-traditional algorithmic topics such as distributed algorithms or cryptography.

Stanford has an introductory Algorithms class with several potential follow-up classes. The most relevant of these classes for my class would be "Advanced Data Structures" (Stanford.edu, 2023). This class contains significant units on range queries, balanced trees, advanced heaps, hashing, and dynamic connectivity.

3. PROPOSED COURSE DESIGN

The Advanced Algorithms course will center around 3 units: *range query data structures, randomized techniques, and string algorithms*. Compared with higher-level algorithms courses at other universities, this class will have fewer units. I have made this choice because the current algorithms class at UVA already covers a reasonably wide breadth of topics. In fact, many of the topics covered in DSA2 are higher level topics at other schools such as max flow or the disjoint set union data structure. In addition, the primary goal for this class would be to prepare students for potential interview questions. Many algorithms taught in higher-level algorithms courses are very niche and are unlikely to appear in an interview. The chosen units for my course would describe general techniques and strategies likely to be used on a wide variety of problems.

3.1 Range Query Data Structures

I have chosen to start the course with a unit on range query data structures. Range query problems are algorithmic problems with multiple repeated queries that ask to compute some function on an interval of a large array. A common example of such a problem would be to compute range sum queries where a program is given an array of integers. Then the program is given several left and right boundaries defining an interval on which the program must return the sum of all elements in the interval. Range queries are included in the class because they cover a wide range of problems and teach students how algorithms can be adjusted to support additional functionality. Last, range queries can be used as an example to demonstrate different classes of algorithmic problems such as online vs. offline and dynamic vs. static.

Initially students would be presented with the static range sum query problem described previously. Students would learn about prefix sum arrays, which solve this problem with O(n) pre-computation and answer a query in O(1). Prefix sum arrays compute the sum of every prefix of an array, a contiguous segment of the array that contains the first element of the array. Then a query interval defined by some left and right indices, L and R, can be computed by returning $prefix(R) - prefix(L-1)$.

The rest of the course will describe modifications to the static range sum query problem. First, the sum function can be replaced by a non-invertible function such as MAX. Students would learn about sparse tables and square root decomposition as potential solutions to this problem. Next, the problem would be modified to be dynamic instead of static. In the dynamic version of the range sum problem, additional queries that update an element in the array must be supported. Students would be taught about segment trees, a divide and conquer data structure, to efficiently handle these queries. Finally, students would learn about offline range mode queries. Range mode queries ask for the most frequent element on the requested interval. Offline queries are given in a batch all at once. This is different than all previous queries which would be given online, meaning that each query response must be given before receiving the subsequent query. To solve this problem, students will learn Mo's algorithm which uses the offline property of the problem to sort the queries in a clever way that allows efficient amortized computation.

3.2 Randomized Techniques

The second unit of the class would be randomized techniques. I chose randomized techniques because it is a very general technique that can be applied to many kinds of problems. In addition, the topic is not covered very well in the current algorithms class at UVA.

To transition from unit 1 to unit 2, students would be presented with another range query problem. The problem would be online range mode queries where the mode only needs to be returned if the mode is a majority of the elements on the queried interval. First, the students would be told a segment tree solution that maintains the most frequent and second most frequent elements within intervals. Afterwards, students would be given a randomized solution where a random element is chosen on the interval. Because the mode only needs to be returned if it is a majority element, if an answer exists, the random guessing will find the answer at worst 50% of the time. If this process is repeated multiple times, the probability of failure will drop exponentially. Guessing around 60 elements will find the mode with negligible probability of failure. The purpose of this problem would be to introduce students to randomized algorithms and demonstrate how randomized algorithms can be significantly simpler relative to their deterministic counterparts.

The next randomized topic would be the birthday paradox. The general statement of this paradox is to imagine a categorization that has N different potential categories. If we randomly choose people who fall in a single category, approximately how many people do we need to take before two of them fall in the same category? The answer to this is $O(\sqrt{N})$ because the number of pairs and thus potential collisions grows at a rate of $O(N^2)$. A motivating algorithmic problem would be to find two strings of a given string that have the same polynomial hash where the hash is computed with a modulus of $10^9 + 7$. Because hashes will be roughly random, choosing around 100000 random strings will almost always find a collision if one exists.

3.3 String Algorithms

I chose string algorithms as a topic for this course because they are a common algorithmic problem type and are not taught thoroughly in the current algorithms class at UVA. We would start with string hashing, which is a review topic from previous classes and was discussed at the end of the randomized unit. Students would be taught useful techniques to manipulate string hashes such as prefix hashes to compute arbitrary substring hash queries.

Next the class would cover the Knuth-Morris-Pratt (KMP) algorithm and Z-function algorithms. These functions allow for efficient string matching queries with very few lines of code. The class would compare string hashing with these algorithms. Hashing is very powerful and can solve many problems with good asymptotic complexity, but it also requires more code and has chances for hash collisions that produce incorrect results.

The last topic in the string algorithm section would be suffix arrays. Suffix arrays efficiently sort the suffixes of a string, which allows for the efficient computation of many counting queries such as counting the number of distinct substrings of a string.

4. PROPOSED COURSE EVALUATION

As the goal of the course is to improve student interview preparation, evaluation for the course would measure its ability to improve interview results. It would be difficult to directly measure this; however, it would be possible to administer a test containing a random set of interview questions at the beginning and end of the course. Although there would be variance in this metric due to the random selection of problems, given a few semesters and enough students, it should give a fairly good representation of the improvement of students through the class. A potential issue of this evaluation would be the potential for students to put very little effort for these tests assuming they are not graded. Potentially, this could be resolved by giving extra credit points based on the test results. However, this may produce further confounding variables where students who have secured an A in the course will not try as hard or not participate in the test at all.

Two other potential evaluation methods would be to ask for student feedback. A similar, more quantitative metric would be the enrollment statistics for the class over several semesters. If enrollment remains high for several semesters and has a long waitlist, it would indicate that the class is doing a good job of improving student algorithmic knowledge. However, there are several reasons for high enrollment, such as instructor reputation or workload, that could invalidate this metric.

Overall, we would want to combine these metrics to gauge the success of the course. Because all of the metrics have noise and confounding variables, it would be important to gather as much data as possible so that the noise can cancel out. Through the combination of data, we would want to see that students have improved algorithmic capabilities after the course.

5. CONCLUSION

As industry requirements for algorithmic programming have increased, UVA has fallen behind in algorithmic preparation of its students. A new Advanced Algorithms course would be able to fill in many of the holes of the current Algorithms course and give UVA students a leg up in coding interviews. My proposed course would focus on range query data structures, randomized techniques, and string algorithms. These topics will supply students with very general techniques that can be used to solve a large variety of problems, especially those found in coding interviews.

6. FUTURE WORK

Many steps can be taken to further strengthen the design of an Advanced Algorithms course. One of the most important is to develop a systematic way for determining the most useful techniques to teach. I chose the current topics in my proposal primarily from personal experience and some research on other schools. Other potential methods could be to search through problem websites or databases for common topics or conduct surveys to ask students what topics they have seen in interviews. It is very important to have a systematic method for determining course topics because course topics must vary with industry needs to best prepare students for their interviews. To do this effectively, we need a consistent method to evaluate the state of industry.

REFERENCES

Stanford.edu. (2023). *Advanced Data Structures Schedule*. CS166: Advanced Data Structures. (2023). https://web.stanford.edu/class/cs166/

codesignal.com. (2023). *CodeSignal's 2023 University Ranking Report*. CodeSignal. (2023, August 16). https://codesignal.com/university-ranking-report-2023/

MIT OpenCourseWare. (2015). *Calendar: Design and analysis of algorithms: Electrical engineering and computer science*. MIT OpenCourseWare. https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/pages/calendar/

cphof.org. (n.d.) *Profile of University of Virginia - Competitive Programming Hall of Fame*. https://cphof.org/university/University%20of%20Virginia