Data Structures Fundamentals at edX: Syllabus

August 15, 2018

Contents

1 Welcome!

Thank you for joining Data Structures Fundamentals at edX! In this course, we consider the common data structures that are used in various computational problems. You will learn how these data structures are implemented in different programming languages and will practice implementing them in our programming assignments. This will help you to understand what is going on inside a particular built-in implementation of a data structure and what to expect from it. You will also learn typical use cases for these data structures.

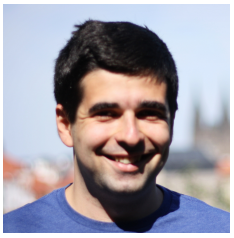
2 Who This Class Is For

Programmers with basic experience looking to understand the practical and conceptual underpinnings of algorithms and data structures, with the goal of becoming more effective software engineers. Computer science students and researchers as well as interdisciplinary students (studying electrical engineering, mathematics, bioinformatics, etc.) aiming to get more profound understanding of algorithms and data structures, and hands-on experience implementing them and applying for real-world problems. Applicants who want to prepare for an interview in a high-tech company.

3 Meet Your Instructors

Daniel Kane is an associate professor at the University of California, San Diego with a joint appointment between the Department of Computer Science and Engineering and the Department of Mathematics. He has diverse interests in mathematics and theoretical computer science, though most of his work fits into the broad categories of number theory, complexity theory, or combinatorics.

Alexander S. Kulikov is a senior research fellow at Steklov Mathematical Institute of the Russian Academy of Sciences, Saint Petersburg, Russia and a lecturer at the Department of Computer Science and Engineering at University of California, San Diego, USA. He also directs the Computer Science Center in Saint Petersburg that provides free advanced computer science courses complementing the standard university curricula. Alexander holds a Ph. D. from Steklov Mathematical Institute. His research interests include algorithms and complexity theory. He co-authored online courses "Data Structures and Algorithms" and "Introduction to Discrete Mathematics for Computer Science".



Michael Levin is a Lecturer at the Computer Science Department of Higher School of Economics, Moscow, Russia and the Chief Data Scientist at the Yandex.Market, Moscow, Russia. He also teaches Algorithms and Data Structures at the Yandex School of Data Analysis.

Neil Rhodes is a lecturer in the Computer Science and Engineering department at the University of California, San Diego and formerly a staff software engineer at Google. Neil holds a B.A. and M.S. in Computer Science from UCSD. He left the Ph.D. program at UCSD to start a company, Palomar Software, and spent fifteen years writing software, books on software development, and designing and teaching programming courses for Apple and Palm. He's taught Algorithms, Machine Learning, Operating Systems, Discrete Mathematics, Automata and Computability Theory, and Software Engineering at UCSD and Harvey Mudd College in Claremont, California.

4 Prerequisites

Basic knowledge of at least one programming language (`C`, `C++`, `Java`, `JavaScript`, `Python2`, `Python3`, `Scala`): loops, arrays, stacks, recursion. Basic knowledge of mathematics: proof by induction, proof by contradiction.

5 Course Overview

Programming Assignments are a very important part of this course. We believe that the only way to understand a data structure is to implement it. If you're studying this course as a part of Algorithms and Data Structures MicroMasters program, you're already familiar with the style of the programming assignments from the first course of the program Algorithmic Design and Techniques. However, if you're taking this class independently, we strongly recommend that before you start working on the Programming Assignments in this course, you first go through the videos and readings of the first week of the Algorithmic Design and Techniques course: they will guide you through the process of implementing a solution, testing it, finding bugs, fixing them and submitting the assignment.

A few examples of questions that we are going to cover in this class are the following:

- What is a good strategy of resizing a dynamic array?
- How priority queues are implemented in C++, Java, and Python?

- How to implement a hash table so that the amortized running time of all operations is O(1) on average?
- What are good strategies to keep a binary tree balanced?

You will also learn what is Block Chain, how does it work, why does it work that way, and what tool does this technology provide in essense.

We look forward to seeing you in this class! We know it will make you a better programmer.

Course Outline

Week 1: Basic Data Structures. In this module, you will learn about the basic data structures used throughout the rest of this course. We start this module by looking in detail at the fundamental building blocks: arrays and linked lists. From there, we build up two important data structures: stacks and queues. Next, we look at trees: examples of how theyre used in Computer Science, how theyre implemented, and the various ways they can be traversed. Once youve completed this module, you will be able to implement any of these data structures, as well as have a solid understanding of the costs of the operations, as well as the tradeoffs involved in using each data structure.

Week 2: Dynamic Arrays and Amortized Analysis. In this module, we discuss Dynamic Arrays: a way of using arrays when it is unknown ahead-of-time how many elements will be needed. Here, we also discuss amortized analysis: a method of determining the amortized cost of an operation over a sequence of operations. Amortized analysis is very often used to analyse performance of algorithms when the straightforward analysis produces unsatisfactory results, but amortized analysis helps to show that the algorithm is actually efficient. It is used both for Dynamic Arrays analysis and will also be used in the end of this course to analyze Splay trees.

Week 3: Priority Queues and Disjoint Sets. We start this module by considering priority queues which are used to efficiently schedule jobs, either in the context of a computer operating system or in real life, to sort huge files, which is the most important building block for any Big Data processing algorithm, and to efficiently compute shortest paths in graphs, which is a topic we will cover in our next course. For this reason, priority queues have built-in implementations in many programming languages, including C++, Java, and Python. We will see that these implementations are based on a beautiful idea of storing a complete binary tree in an array that allows to implement all priority queue methods in just few lines of code. We will then switch to disjoint sets data structure that is used, for example, in dynamic graph connectivity and image processing. We will see again how simple and natural ideas lead to an implementation that is both easy to code and very efficient. By completing this module, you will be able to implement both these data structures efficiently from scratch.

Week 4: Hashing. In this module you will learn about very powerful and widely used technique called hashing. Its applications include implementation of programming languages, file systems, pattern search, distributed key-value storage and many more. You will learn how to implement data structures to store and modify sets of objects and mappings from one type of objects to another one. You will see that naive implementations either consume huge amount of memory or are slow, and then you will learn to implement hash tables that use linear memory and work in O(1) on average! You will also learn what is Block Chain, how does it work, why does it work that way, and what tool does this technology provide in essense.

Weeks 5-6: Binary Search Trees. In this module we study binary search trees, which are a data structure for doing searches on dynamically changing ordered sets. You will learn about many of the difficulties in accomplishing this task and the ways in which we can overcome them. In order to do this you will need to learn the basic structure of binary search trees, how to insert and delete without destroying this structure, and how to ensure that the tree remains balanced. We then study another kind of balanced search trees - Splay Trees. They adapt to the queries dynamically and are optimal in many ways.

6 Learning Objectives

You will learn the fundamental data structures, how do they work, and how to apply them to make your programs run several orders of magnitude faster. In the process, you will implement many of these data structures and apply them in appropriate problems while solving 14 Programming Challenges ranging from basic level to advanced.

7 Estimated Workload

We expect this course will take you 8–10 hours per week to complete.

8 Grading

Your grade will be composed out of the following components: four programming assignments (PA's) and the final proctored exam.

assignment	weight	# problems	# droppable
PA1: Programming Challenges	17%	3	1
PA2: Algorithmic Warm-up	17%	3	1
PA3: Greedy Algorithms	17%	3	1
PA4: Divide-and-Conquer	17%	5	2
Final exam	32%	1	0

The passing thresholds are $\geq 70\%$ for C, $\geq 80\%$ for B, and $\geq 90\%$ for A. To receive a certificate, you need to get at least C. Passing Grade: you must score 70% or above to pass the course. To get a university course credit, you must score at least 80%.

9 Deadlines

This course is self-paced, so there are no deadlines for any specific assignment. This course will be open until April 1, 2019. Shortly after that we expect to re-open the course, self-paced, with updates.

10 Verified Certificate

There are a number of differences for verified learners compared with those just wishing to audit the course.

- **Programming challenges.** Verified learners submit a source code that is then run by the autograder on a set of carefully prepared test cases. Therefore, to solve a programming challenge, a verified learner needs to implement a reliable (that works correctly on all test cases) and efficient (that works in less than one second even on massive datasets) program. Writing fast and correct programs is an important skill of a software engineer. On the contrary, audit learners instead solve a programming quiz where the goal is to submit a result for a single dataset. Hence, for such quizzes, we only check correctness on a single dataset, and we don't check efficiency at all.

- **Private forum threads.** Verified learners have access to private forum threads where they can get help on programming challenges as well as discuss various issues with other verified learners who appreciate the impact of the grade on your potential certificate.

- **Official proof of completion.** By completing the course, verified learners receive an easily shareable official certificate. To earn a verified certificate, you need to be enrolled as part of the verified track, complete identity verification before you take the proctored final exam, and earn a passing grade. If you are auditing the course, you will not receive a certificate.

- **University course credit.** This class is a part of MicroMasters program "Algorithms and Data Structures". By completing the MicroMasters program, you indicate to employers and hiring personnel that you have made a significant investment in completing rigorous, Masters-level courses and content.

  Learners who successfully earn the Algorithms and Data Structures MicroMasters Credential are eligible to apply for admission to the School of Individualized Study (SOIS) Master of Science in Professional Studies at Rochester Institute of Technology.

If a learner applies for admission to the SOIC Master of Science in Professional Studies program at Rochester Institute of Technology, and is accepted, the MicroMasters Credential will count towards 25% of the coursework required by this program.

To receive a MicroMasters certificate, you must receive verified certificates in all eight courses in the program. This course is worth 15 credits. The other courses in the MicroMasters program:

  - Algorithmic Design and Techniques (20 credits)
  - Graph Algorithms (15 credits)
  - NP-Complete Problems (10 credits)
  - String Processing and Pattern Matching Algorithms (10 credits)
  - Dynamic Programming and Its Applications In Genomics and Machine Learning (10 credits)
  - Applications of Graph Algorithms in Genome Sequencing (10 credits)
  - Capstone Project in Genome Assembly (10 credits)

- **Proven motivator to complete the course.** MOOCs data tells that verified learners complete courses at a much higher rate than audit learners.

11 Forum

The forum can be found under the Discussion tab from the course home page. We encourage you to introduce yourself at this forum thread when you enroll into the class. We would be happy to know your name, your background, and your expectations for the class.

We also encourage you to visit the forum each time you work on a programming challenge. We've set up a separate thread for every programming challenge. At this thread, you may ask questions as well as help other learners and learn from other learners.

The instructors and TA's are going to constantly monitor the forums to help the learners to overcome their difficulties as quickly as possible.

Forum Rules

- Follow etiquette rules: respect other learners, make your post valuable for others (in particular, before asking check whether someone else has already asked it; keep your post as short as possible). See more common sense rules to follow.

- Please do not post solutions of programming challenges or their algorithmic parts (this violates the edX honor code), even if they do not pass the tests.

- We encourage you to post starter files for various programming languages. A starter file should only read the input data and write the output data.

- If your first attempt to solve a programming challenge failed, and then you debugged and fixed your program, you may want to share this bug. Examples: "Remember to use // for integer division for `Python3`", "Remember that a class name and a file name should match for `Java`", "Remember to use the `long long` type if you are dealing with large numbers for `C++`", but please do not post the code, just mention the "idea" of the bug.

- You may want also to help other learners by posting small tests that can help to catch a bug. In this case, please comment how did you come up with the test, so that others can learn from your creativity.

- Please note that instructors are not going to reply to the following typical questions.

  - *"My program runs fine on my local machine, but fails in the grader. Could you fix the grader?"*
    All the learners' computers are slightly different, so the only way to make grading fully objective is to grade all solutions on the same server. Sometimes, the result of compiling and running the same program is different on different computers because of different compilers, different compiler settings, different operating system or just some bugs in the program code that lead to undefined behavior. However, it is always possible to write a correct program that works correctly both on your local machine and in our grader. You will have to write such a program to pass. We do our best to specify all the important parameters of the grader that we know of here.
  - *"I use exactly the same compiler and flags as in the grader, but my program has different outputs on my machine and in the grader. Could you check the grader?"*
    This usually happens with buggy programs that run into undefined behavior.
  - *"Any hints about test 42?"*
    Recall that we hide the test cases intentionally. Please test and stress test you program to fix it. If you have already tested a lot (considered all corner cases that you can imagine, constructed a set of manual test cases, applied stress testing), but your program still fails and you are stuck, try to ask for help on the forum. We encourage you to do this by first explaining what kind of corner cases you have already considered (it may happen that when writing such a post you will realize that you missed some corner cases!) and only then asking other learners to give you more ideas for tests cases.
  - *"How to compile/run a starter file?"*
    Please note that though this class has many programming exercises, it is not a programming class. We expect you to know how to program in a programming language of your choice.