String Processing and Pattern Matching Algorithms at edX: Syllabus

August 15, 2018

Contents

1 Welcome!

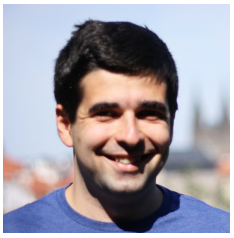
Thank you for joining String Processing and Pattern Matching Algorithms at edX! World and internet is full of textual information. We search for information using textual queries, we read websites, books, e-mails. All those are strings from the point of view of computer science. To make sense of all that information and make search efficient, search engines use many string algorithms. Moreover, the emerging field of personalized medicine uses many search algorithms to find disease-causing mutations in the human genome. In this course, you will learn algorithms and data structures that enable efficient exact and approximate pattern matching, efficient storage of huge strings such as genomes and even efficient pattern matching algorithms working without unpacking the compressed strings!

2 Who This Class Is For

Programmers with basic experience looking to understand the practical and conceptual underpinnings of algorithms, 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 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

Pavel Pevzner is Ronald R. Taylor Professor of Computer Science at the University of California, San Diego. He holds a Ph. D. from Moscow Institute of Physics and Technology, Russia and an Honorary Degree from Simon Fraser University. He is a Howard Hughes Medical Institute Professor (2006), an Association for Computing Machinery Fellow (2010), an International Society for Computational Biology Fellow (2012), and a Member of the the Academia Europaea (2016). He has authored the textbooks Computational Molecular Biology: An Algorithmic Approach (2000), An Introduction to Bioinformatics Algorithms (2004) (jointly with Neil Jones), and Bioinformatics Algorithms: An Active Learning Approach (2014) (jointly with Phillip Compeau). He co-authored online courses "Data Structures and Algorithms", "Bioinformatics", and "Analyze Your Genome!" that are available at Coursera and edX.



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.

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.

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

Course Outline

Week 1: Suffix Trees. How would you search for a longest repeat in a string in LINEAR time? In 1973, Peter Weiner came up with a surprising solution that was based on suffix trees, the key data structure in pattern matching. Computer scientists were so impressed with his algorithm that they called it the Algorithm of the Year. In this lesson, we will explore some key ideas for pattern matching that will - through a series of trials and errors - bring us to suffix trees.

Week 2: Burrows-Wheeler Transform and Suffix Arrays. Although EXACT pattern matching with suffix trees is fast, it is not clear how to use suffix trees for APPROXIMATE pattern matching. In 1994, Michael Burrows and David Wheeler invented an ingenious algorithm for text compression that is now known as Burrows-Wheeler Transform. They knew nothing about genomics, and they could not have

imagined that 15 years later their algorithm will become the workhorse of biologists searching for genomic mutations. But what text compression has to do with pattern matching??? In this lesson you will learn that the fate of an algorithm is often hard to predict its applications may appear in a field that has nothing to do with the original plan of its inventors.

Week 3: Knuth-Morris-Pratt Algorithm. Congratulations, you have now learned the key pattern matching concepts: tries, suffix trees, suffix arrays and even the Burrows-Wheeler transform! However, some of the results Pavel mentioned remain mysterious: e.g., how can we perform exact pattern matching in $O(|Text|)$ time rather than in $O(|Text| \cdot |Pattern|)$ time as in the nave brute force algorithm? How can it be that matching a 1000-nucleotide pattern against the human genome is nearly as fast as matching a 3-nucleotide pattern??? Also, even though Pavel showed how to quickly construct the suffix array given the suffix tree, he has not revealed the magic behind the fast algorithms for the suffix tree construction!In this module, Mihael will address some algorithmic challenges that Pavel tried to hide from you :) such as the Knuth-Morris-Pratt algorithm for exact pattern matching and more efficient algorithms for suffix tree and suffix array construction.

Week 4: Constructing Suffix Arrays and Suffix Trees. In this module we continue studying algorithmic challenges of the string algorithms. You will learn an $O(n \log n)$ algorithm for suffix array construction and a linear time algorithm for construction of suffix tree from a suffix array. You will also implement these algorithms and the Knuth-Morris-Pratt algorithm in the last Programming Assignment in this course.

6 Learning Objectives

You will learn the fundamental data structures for efficient storage of strings and pattern matching, 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 13 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: three programming assignments (PA's) and the final proctored exam.

assignment	weight	# problems	# droppable
PA1: Programming Challenges	25%	5	2
PA2: Algorithmic Warm-up	25%	4	2
PA3: Greedy Algorithms	25%	4	2
Final exam	25%	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 September 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)
  - Data Structures Fundamentals (15 credits)
  - Graph Algorithms (15 credits)
  - NP-Complete Problems (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.