Exploration: Intro to Data Structures

Introduction

To understand the impact of data storage and organization, think for a few minutes about the following problem. Try to come up with a good solution to the problem, and importantly, try also to characterize the way your solution behaves in terms of resource consumption (e.g., how much processing time is needed to perform particular operations in your solution, and how much space is consumed by the structures in your solution as more and more data is added to them).

Imagine you're a developer on a team implementing an exciting web application. One of the most important features of your application is its search functionality. Specifically, at the top of every page within the application is a search box, where users can enter a search query to find content in your application related to that query.

Your team lead has asked you to add an autocomplete feature to this search box. This feature will behave much like Google's autocomplete feature. Specifically, as the user types in the search box, the application will present to them a set of suggested ways to complete the query they're typing, and instead of finishing typing out their whole query, the user can just select one of the suggestions presented to them. If there are many possible suggested completions that match the query being typed, only the top ones will be presented.

The data for this feature is already compiled and provided to you in an alphabetically sorted text file that contains one completion per line.

Now it's up to you to figure out how you are going to store and use that data in your running web application. How will you do it?

Data structures, broadly, are general-purpose mechanisms for storing, organizing, and managing data within a running program.

The term "data structure" also encapsulates the operations associated with a particular structure, for example: the way data is inserted, accessed, and manipulated within the structure.

Importantly, a given data structure represents not only the stored data itself, but also often represents the relationships between specific data elements.

Data Structures is one of the most important courses you'll take as an undergraduate, if you plan to move on to do any kind of programming in your life. Indeed, due to the importance of well-organized data, and because people don't often study data structures when they're learning to program on their own, you can think of this course as to where you move from being a hobbyist

programmer to a budding professional programmer. (And you're likely to get asked about them in interviews!)

Course Overview

In this course, we'll have two primary goals:

- To become familiar with a collection of foundational data structures that you'll use frequently as a programmer, and that will be useful in a wide variety of contexts. These include lists, queues, stacks, trees, hash tables, graphs, and more.
- To understand how to analyze and manage the complexity associated with data structures and their operations. This will allow us to keep our programs' running times and memory usage under control.

None of the data structures we'll see in this class (nor any beyond this class) is a perfect data structure for all situations. Each one involves trade-offs in terms of the following things:

- How long its operations take to run
- How much space it requires to store a collection of data of a given size
- How hard it is to implement

With the understanding you'll gain in this class about data structures and how to analyze them, you'll be able to compare data structures and choose/design the best one for a particular task.

Along the way, we'll also learn a few other things, such as the distinctions between abstract data types (ADTs) and data structures.

- ADTs are data types modeled from the point of view of the user of that data type (e.g., a Python programmer using a dictionary).
    - For example, a hash table can be implemented with either a single list or a collection of lists. The user of the hash table doesn't need to know (or care about) how it's implemented, only how it behaves.
- Data structures are concrete representations of data, i.e., a data type from the point of view of the implementer. The user may not care how the hash table is implemented, but it might make a lot of difference to the developer of that hash table.