CSE 100 Notes

Advanced Data Structures

Fall 2021

Taught by Professor Niema Moshiri

Table of Contents

1 A Brief Introduction

In this course, we will primarily be building off of our prior knowledge of data structures (CSE 12). In particular, we will:

- Analyze data structures for both time and space complexity.
- Describe the strengths and weaknesses of a data structure.
- Implement complex data structures correctly and efficiently.

1.1 Data Structures vs. Abstract Data Types

When talking about data, we often hear about data structures and abstract data types.

Data Structures (DS)	**Abstract Data Type** (ADT)
Data structures are collections that contain: • Data values. • Relationships among the data. • Operations applied to the data. It also describes how the data are organized and how tasks are performed. So, a data structure defines every single detail about anything relating to the data.	Abstract data types are defined primarily by its behavior from the view of the user. So, not necessarily how the operations are done, but rather what operations it must have from a completely abstract point of view. Specifically, it describes only what needs to be done, not how it's done.

Consider the `ArrayList` (DS) vs. the `List` (ADT).

- A `List` will most likely have the following operations:
  - `add`: Adds an element to the list.
  - `find`: Does an element exist in the list?
  - `remove`: Remove an element from the list.
  - `size`: How many elements are in this list?
  - `ordered`: Each element should be ordered in the way we added it. For example, if we added `5`, and *then* added `3`, and *then* added `10`, our list should look like: `[5, 3, 10]`.

  Of course, as an abstract data type, `List` isn't going to define how these operations work. It just lists all operations that any implementing data structure must have. In other words, we can think of `List`, or any abstract data type, as a *blueprint* for future data structures.
- An `ArrayList` is simply an array that is expandable. It is internally backed by an array. So, we can perform the following operations:
  - We can `add` an element to the `ArrayList`. In this case, we add the element to the next available slot in the array, expanding the array if necessary.
  - We can `find` an element in the `ArrayList`. In this case, we can search through each slot of the array until we find the array or we reach the end of the array.
  - We can `remove` an element from the `ArrayList`. In this case, we can simply move every element after the specified element back one slot.
  - We can get the `size` of the `ArrayList`. In this case, this is as simple as seeing how many elements are in this `ArrayList`.
  - And, we know that the `ArrayList` is `ordered`. In this case, this is already done via the `add` and `remove` methods.

Notice how `ArrayList` specifies how each operation defined by `List` works. In this sense, we say that `ArrayList` essentially implements `List` because we need to define *how* the tasks defined by `List` are performed.

So, the key takeaways are:

- An abstract data type (in our case, `List`) specifies what needs to be done without specifying how it's done.
- A data structure (in our case, `ArrayList`) actually defines **how** the data is organized, how the different operations are performed, and how exactly everything is represented.

2 Introduction to C++

Here, we will talk about C++, the programming language that we will use in this course.

2.1 Data Types

First, we'll compare the data types in Java and C++.

Data Type	Java	C++
`byte`	1 byte	1 byte
`short`	2 bytes	2 bytes
`int`	4 bytes	4 bytes
`long`	8 bytes	8 bytes
`long long`		16 bytes
`float`	4 bytes	`4 bytes`
`double`	8 bytes	`8 bytes`
`boolean`	Usually 1 byte	
`bool`		Usually 1 byte
`char`	2 bytes	1 byte

It should be mentioned that:

- In Java, you can only have signed data types.
- In C++, you can have both signed and unsigned data types.
- `boolean` (Java) and `bool` (C++) are effectively the same thing: they represent either `true` or `false`.