19.1 Introduction to Hashing: Data Indexed Arrays

Note: This chapter of the book is based on the Fall 2022 presentation of the material. It will be updated for Spring 2023 at a future date.

So far in the course, we've taken a look at many ways to store things in data structures, but they're not always the most efficient in terms of runtime. Enter an incredible data structure that can provide insertion, removal, and contains checks, regardless of how many elements are inside of it (even if theres millions of items)– all in O(1) runtime in the best case! Sound too good to be true?

We will explore the magic of hashing in this chapter to see how this is possible.

Quick Recap of Data Structures we've seen so far

We've looked at a few data structures that efficiently search for the existence of items within the data structure. We looked at Binary Search Trees, then made them balanced using 2-3 Trees.

However, there are some limitations that these structures impose (yes, even 2-3 trees!)

1. They require that items be comparable. How do you decide where a new item goes in a BST? You have to answer the question "are you smaller than or bigger than the root"? For some objects, this question may make no sense.
2. They give a complexity of $\Theta(logN)$. Is this good? Absolutely. But maybe we can do better.

Professor Hug's Lecture on Hash Tables.

Using Data as Indices

Arrays have amazing runtime for its basic operations. Is there a good way to convert data into indices and store them in an array? In the next couple of sub-sections, we will walk through the steps that will eventually lead us to our final creation -- the hash table. Watching the videos linked in the subsections is helpful for understanding the process through which we arrived at the invention of hash tables. Hash tables will be the main focus of the rest of the chapter.

Previous
19. Hashing I

Next
19.1.1 A first attempt: DataIndexedIntegerSet

Last updated 1 year ago