19.1.1 A first attempt: DataIndexedIntegerSet

Let us begin by considering the following approach. This approach was introduced in Hashing Video 1.

For now, we're only going to try to improve complexity from $\Theta(logN)$ to $\Theta(1)$. We're going to not worry about comparability. In fact, we're going to only consider storing and searching for `int`s.

Here's an idea: let's create an ArrayList of type `boolean` and size 2 billion. Let everything be false by default.

- The `add(int x)` method simply sets the `x` position in our ArrayList to true. This takes $\Theta(1)$ time.
- The `contains(int x)` method simply returns whether the `x` position in our ArrayList is `true` or `false`. This also takes $\Theta(1)$ time!

```
public class DataIndexedIntegerSet {
    private boolean[] present;

    public DataIndexedIntegerSet() {
        present = new boolean[2000000000];
    }

    public void add(int x) {
        present[i] = true;
    }

    public boolean contains(int x) {
        return present[i];
    }
}
```

There we have it. That's all folks.

Well, not really. What are some potential **issues** with this approach?

- Extremely wasteful. If we assume that a `boolean` takes 1 byte to store, the above needs `2GB` of space per `new DataIndexedIntegerSet()`. Moreover, the user may only insert a handful of items...
- What do we do if someone wants to insert a `String`? Or other data types?
  - Let's look at this next. Of course, we may want to insert other things, like `Dog`s. That'll come soon!

Previous
19.1 Introduction to Hashing: Data Indexed Arrays

Next
19.1.2 A second attempt: DataIndexedWordSet

Last updated 1 year ago