CISC-235

20200109

In most of our discussions of data structures, the values being stored will simply be integers. It is important to recognize that in actual applications, data items usually consist of a collection of attributes of an object (for example, all the information regarding a book in a library, or all the information about a particular pharmaceutical product). Normally one attribute is recognized as being an unique identifier of the object (for example, the ISBN of a book) – we usually call this the key. This is what is represented by the simple integers that we will store in our structures. We will not often concern ourselves with the other attributes of the data objects, but we should keep their existence in the back of our minds.

Since we now recognize that in order to decide on the best representation for our data we need to know what operations we will be performing, it behooves us to consider some of the common operations on a set of values or items. These can be grouped into 2 categories:

Operations related to a single item:

- add an item to the set
- remove an item from the set (these first two operations do not necessarily go together: some data sets never grow, and some never shrink)
- attach extra data to an item (such as adding a new email address to a person in your contact set)
- find the successor of an item (that is, find the one that would come next in sorted order)
- find the predecessor of an item
- search for a particular item (this has two forms: "Is x in the set?" and "What is x's location in the set?")

Operations related to the entire set:

- combine two (or more) sets into a single set (A brief excursion into this topic, to illustrate how the choice of data structure can affect the amount of work required to perform an operation. If the sets are stored in arrays, combining two sets requires

copying the items in one array to empty spaces in the other array, if possible. If there is insufficient empty space, a new array must be allocated and all the items must be copied into the new array. On the other hand if the sets are stored in linked lists, the combined set is created by appending one list to the end of the other, which is accomplished in a single step.

- sort the set
- find the max and/or min element in the set
- perform a range query on the set (find all elements x such that a $\leq$ x $\leq$ b, for some specified a and b values)
- find a subset of the set based on one or more attributes (such as "find all red sports cars in the "Vehicles for sale" set)

One goal of this course is to give you an understanding of many of the most important data structures and their strengths and weaknesses, so that when implementing an algorithm you can choose a data structure that is well-suited to the problem.

What criterion should we use to choose an appropriate data structure for an application?

How about "I already understand data structure A, and I don't understand data structure B"? .... umm, no.

Perhaps "There is a built-in module for data structure A, but I would have to code data structure B myself"? .... fail!

Or “I can code A in 5 minutes, but B would take an hour” ... nope, that’s not a good reason.

What could be left? What could be right?

The answer is **computational complexity.** We will prefer structure A to structure B if A has a lower order of complexity for the operations we need in our particular application.

Before we discuss computational complexity, we need to clarify which operations can be completed in constant time.

We assume that all fundamental operations:

- +, -, *, / and comparisons for integers and floating point numbers
- comparisons on Booleans
- comparisons and type conversions on characters
- execution control
- accessing a memory address
- assigning a value to a variable

take constant time

It is important to note that this model implies an upper limit on the number of digits in any number. This is true of virtually all programming languages.

This model does not assume constant time operations on strings. A string is considered to be a data structure consisting of a sequence of characters.

I expect we have all seen "big O" complexity classification (since it has been covered in other courses), but we will review the ideas anyway, starting in the next class.