CISC-235

Data Structures

Winter 2020

We started with this question: How would you complete this equation:

Programs = _________ + __________

Niklaus Wirth, one of the first researchers into good program design (and the inventor of programming languages such as Algol W, Pascal and Modula-2) used a version of this equation as the title of his important book:

Algorithms + Data Structures = Programs

The book was later revised and republished as "Algorithms and Data Structures" It is a great book on the subject and well worth your attention. It is available as a free PDF – the link is given in the Recommended Readings section of the course website.

I read the class a few quotes from this book. These are all from Section 1.1 of "Algorithms and Data Structures" by Niklaus Wirth, 1985:

"The modern digital computer was invented and intended as a device that should facilitate and speed up complicated and time-consuming computations. In the majority of applications its capability to store and access large amounts of information plays the dominant part and is considered to be its primary characteristic, and its ability to compute, i.e. to calculate, to perform arithmetic, has in many cases become almost irrelevant."

"In solving a problem with or without a computer it is necessary to choose an abstraction of reality, i.e. to define a set of data that is to represent the real situation. ... Then follows a choice of representation of this information."

"The choice of representation of data is often a fairly difficult one, and it is not uniquely determined by the facilities available. It must always be taken in the light of the operations that are to be performed on the data."

The points made in these three quotes are even more true now than they were in 1985.

As an example, Wirth describes how we can choose different representations for integers depending on what range of values we are using and what we want to do with them. If we are only using small positive integers and the only operation we need to do is add two numbers, then the best representation is to use "unary" or "base-1" notation in which the number n is just represented by a collection of n marks (or n pebbles if paper is not available).

Suppose we use ☎ as the mark (I tried to find a symbol that would suggest the prehistoric origin of this form of arithmetic – the rotary dial phone seemed appropriate). The the number we usually represent as 6 would be written as ☎☎☎☎☎☎ and the number we usually represent as 5 would be written as ☎☎☎☎☎ Now adding these numbers is trivially easy – we just write their representations together to get the representation of their sum: ☎☎☎☎☎☎☎☎☎☎☎

We can even do multiplication using this representation. For example to multiply ☎☎☎☎ by ☎☎☎ we can create a table like this

	☎	☎	☎
☎	🟦	🟦	🟦
☎	🟦	🟦	🟦
☎	🟦	🟦	🟦
☎	🟦	🟦	🟦

And fill it in:

	☎	☎	☎
☎	☎	☎	☎
☎	☎	☎	☎
☎	☎	☎	☎
☎	☎	☎	☎

And then copy the marks we used to fill in the table: ☎☎☎☎☎☎☎☎☎☎☎☎

Compare this to doing the same operations using the Indo-Arabic numerals we are familiar with. To compute 6+5 and 4*3 we have to learn (and/or memorize) a set of rules that involve how the digits combine with each other, how to "carry the 1" and so on. The base-1 representation for this application is much simpler.

But now suppose we need to work with large integers. The base-1 representation method is not feasible for adding or multiplying numbers in the trillions ... whereas the methods we know for carrying out these operations when the numbers are represented in base-10 notation are fast.

This is an important point that we will return to later. Sometimes we prefer different representations for our data when the data is "small" compared to when it is "large" ... with those terms in quotation marks because sometimes they are not precisely defined.

Turning to the subject of data structures, we considered a few problems:

1. How would you organize a dictionary to facilitate auto-complete and/or spell-checking?

2. How would you represent the structure of the Internet so as to efficiently determine good routes for email?

3. How would you efficiently store a list of "forbidden" words, so that a document could be quickly tested to see if it contains any word in the list, if you were willing to have a small percentage of documents falsely reported as containing forbidden words? Equivalently, how would you store a list of existing user-names so that when a new user selects a user-name you can quickly determine if it has already been taken - again, a small number of "false positive" results is acceptable.

4. Suppose you had a data set which was so large that it had to be stored on an external hard drive, so that accessing the data is significantly slowed by the speed of communication between your computer and the external storage device. How would you organize the data on the external drive to optimize the process of accessing all elements of the set that meet certain criteria (for example, if the data represents students we might want to access all students whose student numbers begin with "11")?

5. How would you create a secure login system with usernames and passwords in which the passwords are not stored, even in an encrypted form, anywhere?

These problems all have two things in common: each one

- involves organizing information
- has a particular goal

This illustrates the first and crucial point about choosing a data structure to organize a set of information - we need to know what we want to do with it. Some data structures are particularly suited for searching the set, while others are optimal for adding and deleting members of the set, or finding subsets, etc. Thus we need to think about the types of operations we may need to implement for a data set, since these will definitely influence our decision.