Lecture 17: Recursive Thinking

The next part of the course introduces a fundamental concept in computing known as *recursion*. Recursion is a phenomenon that spans math, computer science, and in fact, all of nature. We have all seen the self-similar "fractal" structure of a snowflake:

Figure 1: Snowflake

but less obvious examples of recursion occur even in other kinds of geometric shapes, such as seashells:

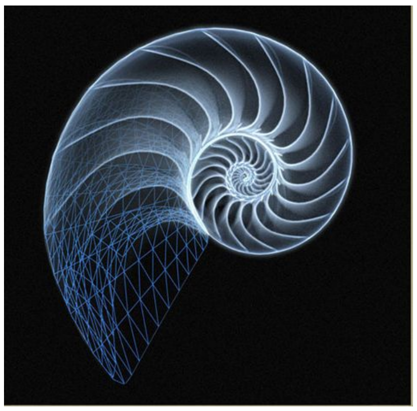

Figure 2: Sea Shells

The reason why recursion is so powerful is that it captures a rather simple way to:

> Build complex structures out of very simple building blocks.

In what follows, we will build a theoretical framework for setting up recursions (or recurrences), understand how to solve recursions, and develop a powerful technique called *mathematical induction* that can be used to prove theorems about recursive relations (among several other things).

Sequences, summations, and how to model them as recursions

We are regularly used to thinking about sequences of objects. Imagine the non-negative integers:

0, 1, 2, 3, 4, 5, ...

or non-negative odd numbers

1, 3, 5, 7, 9, ...

It is not fun to exhaustively enumerate very long (or infinite) sequences of the form:

$a_0, a_1, a_2, a_3, \ldots$

So a more compact way to represent them as a *recurrence relation*. Consider the non-negative integers. One can recursively define them using a two-step procedure. First, write down:

$$a_0 = 0,$$

and then define every other element of the sequence as:

$$a_n = a_{n-1} + 1.$$

On the other hand, the odd numbers can be defined using the recursion:

$$a_0 = 1,$$
$$a_n = a_{n-1} + 2.$$

The first specification ($a_0 = 1$) is called the *base case*, and the second is called the *recursive step*.

More generally, sequences of numbers (integers or reals) defined via a recursion of the form:

$$a_0 = b,$$
$$a_n = a_{n-1} + d.$$

are called *arithmetic progressions*. Observe that an arithmetic progression is any sequence of regularly-spaced numbers of the form $(b, b + d, b + 2d, b + 3d, \ldots)$.

Likewise, a different example. Suppose we invest b dollars in a bank, where the (monthly) appreciation ratio is r (i.e., at the end of each month the value gets multiplied by a factor r). In that case, a recursive relation for the amount of money in our account at the end of the nth month is given by:

$$a_0 = b,$$
$$a_n = r a_{n-1}.$$

Such a sequence provides numbers of the form:

$b, b \cdot r, b \cdot r^2, b \cdot r^3, \ldots$

which is called a *geometric progression*.

The situation changes if we are a bit more fiscally prudent, and invest a fixed amount of money *each* month. In that case, the balance in the account at the n^{th} month, a_n, is given by:

$$\begin{aligned} a_0 &= b, \\ a_n &= ra_{n-1} + b. \end{aligned}$$

Recursions and closed form expressions

In the above examples, we have discussed how to set up a recursion. However, it is convenient to find a *closed-form* expression for the nth element of the recursion. Solving a recursion involves two steps: 1) guessing a *closed-form* expression and 2) *proving* that this is correct.

Let us go back to the example of an arithmetic progression defined above.How do we compute the n^th element of this sequence? The first way is to start at $a_0 = b$, obtain $a_1 = b + d$, $a_2 = a_1 + d = b + 2d$ and so on, until we apply the recursion n times. This is called *forward substitution*. A small amount of pattern matching lets us guess that the n^{th} element of the sequence is given by:

$$a_n = b + dn.$$

Such a formula is a *closed-form expression*: it explicitly encodes a_n as terms of a function of n. Using such a closed-form expression, one can compute a_n for any arbitrary n without enumerating all intermediate elements in the sequence.

Similarly, consider a geometric progression (numbers of the form $b, rb, r^2b, \ldots$), defined using the recursion:

$$\begin{aligned} a_0 &= b, \\ a_n &= a_{n-1}r. \end{aligned}$$

To compute a closed-form expression for a_n, we can use forward substitution, as defined above. However, let's now try a slightly different approach. Start with a_n, and repeatedly invoke the recursion, i.e.,

$$\begin{aligned} a_n &= a_{n-1}r. \\ &= a_{n-2} \cdot r \cdot r \\ &= a_{n-3} \cdot r \cdot r \cdot r \\ &\ldots \\ &= a_0 \overbrace{r \cdots r}^{\text{n times}} \\ &= br^n. \end{aligned}$$

This method is called *backward substitution*.

Before we conclude this section, here is an **exercise**. Can you model the following sequences using a recursion, and if possible, guess a closed form expression?

$3, 9, 27, 81, 327, \ldots$

$1, 2, 6, 24, 120, 720, \ldots$

$1, 1, 2, 3, 5, 8, \ldots$

(The last sequence has a special name: it is called a *Fibonacci sequence*, and has some fascinating and surprising applications. Look it up!)

Summations

Consider any sequence $(a_0, a_1, a_2, \ldots, a_n)$. The *summation* of this sequence is given by:

$$S_n = a_0 + a_1 + \ldots a_n \quad = \sum_{i=0}^{n} a_i.$$

The lower and upper limits (below and above the Σ respectively) indicate the set of elements that participate in the summation.

Important note #0: any summation S_n itself is a sequence. (Observe that the value of S_n depends on n, which is an integer.)

Important note #1: any summation S_n can be defined using recursion! Here is the setup; the logic should be clear from the (literal) definition of summation.

$$\begin{aligned} S_0 &= a_0 \\ S_n &= S_{n-1} + a_n. \end{aligned}$$

Here are a few examples of summations.

$$\begin{aligned} \sum_{i=0}^{n} i &= 1 + 2 + \ldots + n. \\ \sum_{i=0}^{4} i^2 &= 0 + 1 + 4 + 9 + 16 = 30. \\ \sum_{i=0}^{3} (-1)^i &= 1 + (-1) + 1 + (-1) = 0. \\ \sum_{i=3}^{73} i^3 &= 3^3 + 4^3 + 5^3 + 6^3 + \ldots 73^3. \end{aligned}$$

Important note #2: the limits can be shifted using a change of variable. For example, in the last summation above, if $i = j + 3$, then we can plug in the counter j instead of i to obtain the (completely equivalent) summation:

$$\sum_{j=0}^{70} (j+3)^3 = 3^3 + 4^3 + 5^3 + 6^3 + \ldots 73^3.$$

Note that the upper and lower limits have now changed; be mindful of this whenever you do a change of variables while doing a summation.

We can derive *closed* form expressions for some common summations. Consider the special case where the base is zero and the step-size is 1, i.e., the sequence is the natural numbers $(1, 2, 3, 4, 5, \ldots)$. Then, the sum of the arithmetic series is given by:

$$S_n = 1 + 2 + \ldots + n - 1 + n.$$

There is a particularly elegant way to simplify this summation. Trivially, one can observe that this is the same as

$$S_n = n + n - 1 + \ldots + 2 + 1.$$

Now, we will add the two equations term-by-term. We get:

$$2S_n = (n+1) + (n+1) + (n+1) + \ldots (n+1) = n(n+1)$$

or, solving for S_n, we get

$$S_n = \frac{n(n+1)}{2}.$$

For general arithmetic progressions $(a, a+d, a+2d, \ldots,)$, one can show (as an **exercise**) that

$$\begin{aligned} S_n &= a + (a+d) + \ldots + (a+nd) \\ &= \frac{n+1}{2}(2a+nd). \end{aligned}$$

(Aside: The sequence S_n is called the sequence of *triangular* numbers. A possibly apocryphal story is that the legendary mathematician Carl Friedrich Gauss figured out this derivation for S_n

. . .

when he was in primary school.

So there's that.)