Recursions

Victor Adamchik

Fall of 2005

Plan

Forms of Recursion

In the previous lecture we have formally introduced recursion and recursive definition. It has been noted that recursion has a lot in common with induction. However, you have to understand the main difference between them. Induction is a method of proof. Recursion is a method of construction (defining). The simplest example of recursion is a primitive recursion which is a way of defining $f(n+1)$ in terms of $f(n)$. All primitive recursive functions are *Turing computable*. The simplest examples of primitive recursive functions are sum, product, factorial and exponentiation over nonnegative integers

```
int sum(int x, int y) {
  if (x==0 || y==0) return x+y;
  else return 1 + sum(x-1, y); }

int product(int x, int y) {
  if (x==0 || y==0) return 0;
  else return y + product(x-1, y); }
```

In the following sections we consider the following forms of recursion

recursion with parameters
course-of-value recursion
double recursion
iterations

■ Recursion with Parameters

■ The Towers of Hanoi

You all know the game Towers of Hanoi. You have three pegs, and a collection of disks of different sizes. Initially, all disks are stacked up at peg 1, the largest disk at the bottom, then the next largest, and so on, up to the smallest disk which sits on top. A move in this game consists of moving the top disk from one peg to another, subject to the condition that a larger disk may never come to rest on a smaller one. The objective is to find a sequence of admissible moves that will bring all the disks to peg number 2. The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. The puzzle is well known to students of Computer Science since it appears in virtually all introductory text on data structures and discrete mathematics.

A couple lectures back we proved (by induction) that tt takes $2^n - 1$ moves to move n disks from the first peg to the third peg. Here we develop a recursive solution.

Let $T(n, 1, 3)$ be the minimum number of moves needed to solve the puzzle with n disks. We will derive a recursive formula for $T(N, 1, 3)$. We break down the problem into three steps:

1. move $(n - 1)$ disks from 1 to 2
2. move one disk from 1 to 3
3. move $(n - 1)$ disks from 2 to 3

#1 peg : n disks	**#1 peg :** 1 disk	**#1 peg :** 0 disk	**#1 peg :** 0 disk
#2 peg : 0 disk	**#2 peg :** $n - 1$ disks	**#2 peg :** $n - 1$ disks	**#2 peg :** 0 disk
#3 peg : 0 disk	**#3 peg :** 0 disk	**#3 peg :** 1 disk	**#3 peg :** n disks

Therefore, a recursive solution is given by:

$$T(n, \mathbf{1}, \mathbf{3}) = T(n - 1, \mathbf{1}, \mathbf{2}) + T(1, \mathbf{1}, \mathbf{3}) + T(n - 1, \mathbf{2}, \mathbf{1})$$

Such recursion is called a recursion with parameters. If we ignore parameters we get the following equation for number of moves:

$$T(n) = 2\,T(n - 1) + 1$$

Course-of-value Recursion

The Fibonacci Numbers

Fibonacci was born 1170 in Pisa, Italy and died in 1250. His real name is Leonardo Pisano. In 1202 he wrote a book: *Liber Abbaci*, meaning "Book of Calculating". It helped bring arithmetic in Europe up to Arab standards. One of Fibonacci's accomplishments was to determine how rabbits reproduce:

A man has one pair of rabbits at a certain place entirely surrounded by a wall. We wish to know how many pairs can be bred from it in one year, if the nature of these rabbits is such that they breed every month one pair (male and female), that in turn will begin to breed in the second month after their birth.

month	repro.	non-repro.	total
1	0	1	1
2	1	0	2
3	2	1	3
4	3	2	5
5	5	3	8

This provides the definition for Fibonacci numbers

$$F_{n+2} = F_{n+1} + F_n$$

$$F_0 = 0; F_1 = 1$$

Note, we use two values F_n and F_{n+1} in the definition. Such a recurion is called a course-of-value recursion. Course-of-value recursion allows the use of any number of values for previous arguments.

Lucas numbers are the companions to the Fibonacci numbers and satisfy the same recurrence by different initial conditions

$$L_{n+2} = L_{n+1} + L_n$$

$$L_0 = 2; L_1 = 1$$

N-bit Strings

Let S_n denotes the number of n-bit strings that do not contain the pattern 111. Develop the recurent formula for S_n.

Consider three cases

1) strings begin with 0
2) strings begin with 10
3) strings begin with 11

Suppose an n-bit string begins with 0 and does not contain 111. Then by removing the first bit we get the $(n-1)$-bit string that does not contain 111. Therefore, there are S_{n-1} such strings.

Suppose an n-bit string begins with 10 and does not contain 111. Then the $(n-2)$-bit string following 10 and does not contain 111. Therefore, there are S_{n-2} such strings.

Suppose an n-bit string begins with 11 and does not contain 111. The third bit must be 0. Then the $(n-3)$-bit string following 110 and does not contain 111. Therefore, there are S_{n-3} such strings.

Thus,

$$S_n = S_{n-1} + S_{n-2} + S_{n-3}$$

Initial conditions

$$S_1 = 2$$
$$S_2 = 4$$
$$S_3 = 7$$

■ Double Recursion

■ The Archimedes Number

Primitive recursion is used to define functions of one variable but with one or more parameters. Double recursion allows the recursion to run on two variables.

Let us consider so-called the Archimedes problem: *I will try to show you by means of geometrical proofs, which you will be able to follow, that, of the numbers named by me <...>, some exceed not only the number of the mass of sand equal in magnitude to the earth filled up in the way described, but also that of a mass equal in magnitude to the universe.* Archimedes defines his number recursively as

$$h(0, n) = 1$$
$$h(m+1, 0) = h(m, a)$$
$$h(m+1, n+1) = a * h(m+1, n)$$

Solving the double recurence (assuking that $a = 10^8$), Archimedes gets an estimate of 10^{63} for the number of grains of sand needed to fill the universe.

Let us unwind a particular cases when $m = 0$:

$$h(1, 0) = 1$$
$$h(1, n+1) = a * h(1, n)$$

The first argument of h is a parameter, therefore, by denoting $h(1, n) = h^*(n)$ we get

$$h^*(0) = 1$$
$$h^*(n+1) = a * h^*(n)$$

This suggests that Archimedes' double recursion is reducible to primitive recursion. In the next section we consider double recursion that is not reducible.

■ The Ackermann Function

This is the famous Ackermann function that grows faster that any primitive recursive function

$$A(0, n) = n + 1$$
$$A(m+1, 0) = A(m, 1)$$
$$A(m+1, n+1) = A(m, A(m+1, n))$$

Let us unwind a few particular cases.
Case $m = 0$:

$$A(1, 0) = 2$$
$$A(1, n+1) = A(1, n) + 1$$

which has solution $A(1, n) = n + 2$.
Case $m = 1$:

$$A(2, 0) = 3$$
$$A(2, n+1) = A(2, n) + 2$$

which has solution $A(2, n) = 2n + 3$.
Case $m = 2$:

$$A(3, 0) = 5$$
$$A(3, n+1) = 2\,A(3, n) + 3$$

which has solution $A(3, n) = 2^{n+3} - 3$.
Case $m = 3$:

$$A(4, 0) = 13$$
$$A(4, n+1) = 2^{A(4,n)+3} - 3$$

which has solution $A(4, n) = 2^{2^{\cdot^{\cdot^{2}}}} - 3$ (exponentiation is $n+3$ times). To see how big that number, we compute it for $n = 2$,

$$A(4, 2) = 2^{2^{2^{2}}} - 3 = 2^{65536} - 3$$

Regardless of its astronomical growth, Ackermann's function is Turing computable.

■ Iterations

■ Infinite Continued Fractions

We use the Euclidean Algorithm to find a GCD of two integers a and b

$$\begin{aligned} a &= b\,q_1 + r_1, \\ b &= r_1\,q_2 + r_2, \\ r_1 &= r_2\,q_3 + r_3, \\ &\cdots \\ r_{k-2} &= r_{k-1}\,q_k + r_k, \\ r_{k-1} &= r_k\,q_{k+1} + 0, \end{aligned}$$

Eliminating remainders from the system, yeilds a continued fraction

$$\frac{a}{b} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{q_4 + \ldots + \cfrac{1}{q_{k+1}}}}}$$

One can easily demonstrates that any rational number has a continued fraction representation. But about algebraic or transcendental numbers? Let us apply the Euclidean Algorithm to real numbers. Compute GCD$\left(\sqrt{2},\ 1\right)$:

$$\begin{aligned} 1.414214 &= 1 * 1 + 0.414214 \\ 1 &= 2 * 0.414214 + 0.171573 \\ 0.414214 &= 2 * 0.171573 + 0.071068 \\ 0.171573 &= 2 * 0.071068 + 0.029437 \\ 0.071068 &= 2 * 0.029437 + 0.012193 \\ &\ldots \text{ and so on} \end{aligned}$$

This process yeilds the infinite continued fraction for $\sqrt{2}$:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2+\cdots}}}}}$$

This can be rewritten in the following form

$$x_{n+1} = \frac{1}{2 + x_n}$$
$$x_0 = 0$$

where the x_n sequence approaches $\sqrt{2} - 1$ when $n \to \infty$.

Exercise. Consider the following recurrence

$$x_{n+1} = \frac{1}{1 + x_n}$$
$$x_0 = 0$$

What does x_n converge to when $n \to \infty$?

The Newton Iteration

Newton's method is a process of finding an accurate root of a single equations $f(x) = 0$. We suppose that f is a differentiable function on a given interval and then use two terms of Taylor's expansion near x

$$f(x) = f(x_0) + f'(x_0)\,(x - x_0)$$

Taking into account that $f(x) = 0$ and dividing both sides by $f'(x)$, we obtain

$$0 = \frac{f(x_0)}{f'(x_0)} + (x - x_0)$$
$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

The Newton method is given then by the following recurrence

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
$$x_0 = \textit{any number}$$

In particular, this can be used to find a square root of any positive number $x^2 - c = 0$

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{c}{x_n}\right)$$
$$x_0 = 1$$