Analysis of Algorithms: Recursion Equations

Author: Haldir[RET]

Abstract

Keywords: Math, Altihmetics, Recursion, Modulo, Algorithm

There isn't much you need to know to understand this Essay, you should be familiar with mathematical symbols and what Modulo is. The library used in this essay is: NTL by Victor Shoup (http://shoup.net/ntl), since it's very useful for calculations on the integer ring, modular arithmetics are part of it. In the source code below, ZZ is the representation of the integer ring and ZZ_p of the integer ring modulo p. A short explanation to $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, floor(n/2) means that the floating point part of the number is rounded off and ceiling(n/2) is rounded up.

1 What are Recursion Equations and what types exist

Recursion Equations are equations, which are recursively defined. Example:

```
__int64 result = 7;
while(true) {
result *=5;
result %= 0x100000000;}
```

There are two types of Recursion Equations, the first are the homogeneous equations, without a linear factor ("offset"), the second are the inhomogeneous with the factor. You can create equations of different order for each type, order just means how many previous results are used.

2 The linear, homogeneous Recursion Equation of first order

Our first example was the most basic form, the linear,homogeneous Recursion Equation of first order:

$$Definition\ 1:\ x_n = ax_{n-1}\ for\ all\ n \geq 1\ and\ x_0 = b_0$$

What's recursive about it ? The result from the previous round of the loop is used in the next round etc. Let's write it down mathematically, with modulo since we work in a 32Bit register: $x_n = 5x_{n-1}\ modulo\ 2^{32}$, with $x_0 = 7$. Now it would be rather annoying, if we need to run that loop 456789 times, just to get x_{456789}, let's see how we can simplify that, it's rather easy to see that the equation which solves that directly is:

$$x_n = x_0 a^n$$

In our case $x_n = 7 \cdot 5^n\ modulo\ 2^{32}$ the solution would be 892687795.

3 The linear, inhomogeneous Recursion Equation of first order

It is more or less the same as the previous type just with an offset. Solving it is more complex though:

$$Definition\ 2:\ x_n = ax_{n-1} + b_1\ for\ all\ n \geq 1\ and\ x_0 = b_0$$

Now the solution is :

$$x_n = \begin{cases} b_0 a^n + b_1 \frac{(a^n-1)}{a-1} & \text{if } a \neq 1, \\ b_0 + n b_1 & \text{if } a = 1. \end{cases}$$

Example: Let's take some basic linear congruence generator (LCG)

$$x_n = 16807\, x_{n-1} + 78125\ modulo\ \left(2^{31} - 1\right)$$

Source code might look like:

```
__int64 result  = 12345;
while(true) {
result = 16807*result+78125;
result %= 2147483647 }// this is (2^31) -1 (prime)
```

This is one of the most basic LCG types, the early rand() versions used similar ones, even without b_1, see Knuth's book for more details. Now let's see how we can get b_0, b_1 and a. Maybe you have some consecutive numbers and a good guess (it's worth to try anyway) that they are LCG output,but you don't have the actual LCG routine. Three variables, so we need three equations, let's take three consecutive numbers, the first three in this example:

$$x_1 = 207560540$$

$$x_2 = 956631177$$

$$x_3 = 2037688522$$

Some short NTL program which solves that: (Maple simplified the equations)

```
void revlcg()
{
  ZZ p;
  p = 2147483647;
  ZZ_p::init(p);
  ZZ_p x1,x2,x3,a,b0,b1;
  x1 = 207560540;
  x2 = 956631177;
  x3 = 2037688522;

  a = (x3-x2)/(x2-x1);
  cout << a << "\n";
  b0 = (x1*x3+x1*x2-(x2*x2)-(x1*x1))/(x3-x2);
  cout << b0 << "\n";
  b1 = -(x1*x3-(x2*x2))/(x2-x1);
  cout << b1 << "\n";
}

output is: 16807 12345 78125
```

If you don't know the modulo from the LCG I recommend you bruteforce over the primes from the largest number you have, to datatype size limit and take one or two more values to check the bruteforce. NTL's NextPrime function might help you with this. (in our example from 2037688523 to 2^{32}), since it would complicate the equations extremely if you try to solve it generically for all moduli. That's it about the second type of recursive functions,now you know why these simple functions are bad pseudo random number generators. You can extend both types to an higher order, I will discuss the most famous of these higher order equations in the next part, usually solving higher order Recursive Equations needs a Computer Algebra System like Maple, Mathematica etc.

4 The linear, homogeneous Recursion Equation of second order

The second order adds another result from a previous round of the loop to the equation.

$$Definition\ 3: \ x_n = a_1 x_{n-1} + a_2 x_{n-2} \ for \ all \ n \geq 2 \ and \ x_1 = b_1, x0 = b_0$$

Now solving that gets even more complex, usually you don't want to do that manually. Let alpha and beta be two real number solution, which is a floating point number, of the Equation $t^2 - a_1 t - a_2 = 0$ and:

$$A := \begin{cases} \frac{b_1 - b_0\beta}{\alpha - \beta} & \text{if } \alpha \neq \beta, \\ \frac{b_1 - b_0\alpha}{\alpha} & \text{if } \alpha = \beta. \end{cases} \quad B := \begin{cases} \frac{b_1 - b_0\alpha}{\alpha - \beta} & \text{if } \alpha \neq \beta, \\ b_0 & \text{if } \alpha = \beta. \end{cases}$$

Then the solutions to the linear, homogeneous Recursion Equation of second order is:

$$x_n = \begin{cases} A\alpha^n - B\beta^n & \text{if } \alpha \neq \beta, \\ (An + B)\,\alpha^n & \text{if } \alpha = \beta. \end{cases}$$

Now what about that most famous second order equation ?
It's the Fibonacci Numbers Equation.

$$F_n = F_{n-1} + F_{n-2} \ for \ all \ n \geq 2 \ and \ F_1 = 1, F_0 = 0$$

The first few Fibonacci Numbers are: 0,1,1,2,3,5,8,13...
Without Recursion the Fibonacci Numbers would look like this:

$$F_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

The numbers $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$ are called Golden Section (Golden Ratio) and it's a common value for natural proportions. Usually linear Recursion Equation of second order appear if you analyze Algorithms which use Divide and Conquer (you process parts of the problem and combine the individual results). As an example the number of compares in MergeSort would be

$$C_n = C_{\lfloor n/2 \rfloor} + C_{\lceil n/2 \rceil} + n \ for \ all \ n > 1 \ and \ C_1 = 0$$

This is an inhomogeneous equation and leads to the so called "Master Theorem", which discusses the running time of these Algorithms, but the Master Theorem would honestly go beyond the scope of this essay. Now you should have a fairly good overview about Recursive Equations and how to solve them, including a small peek into the Analysis of LCG.

$$Haldir[RET]$$