Using Java to Convert Decimal Fractions to Binary Fractions with Arbitrary Accuracy

Connor Baker

January 2017
Version 0.2d

Abstract

This paper briefly discusses some of the challenges faced with converting a decimal fraction into its binary representation, creating a Java implementation to do so, and writing it in such a way that we can have arbitrary precision. Use of the traditional algorithm for converting decimal fractions to binary fractions is derived, as well as several faster algorithms that exploit powers of two.

Contents

1 A Brief Explanation of the Problem

Given a decimal fraction (that is to say, a fraction in base ten written without the division operator, like 0.5 instead of 1/2), how can one convert it to the equivalent binary representation?

To begin dissecting this problem, I resolved to first look at how conversion from base ten to base two with whole numbers worked.

1.1 From Decimal to Binary

Given a number $ABCDE$ in any base β, one can rewrite that number as the sum of each digit multiplied by the base to a power, like so: $A \times \beta^4 + B \times \beta^3 + C \times \beta^2 + D \times \beta^1 + E \times \beta^0$. This is called exponential form. As an example, given the number 44675_{10} (the notation means 44675 in base ten), we can rewrite it as $4 \times 10^4 + 4 \times 10^3 + 6 \times 10^2 + 7 \times 10^1 + 5 \times 10^0$.

In terms of changing the base of a number, the goal is to rewriting it as the sum of some numbers multiplied by the desired base to a power. Take for example the number 7_{10}. We know that we can rewrite it as 7×10^0, but how could we express this number in base two? Our goal is to end up with something that looks like $a_1 \times 2^n + a_2 \times 2^{n-1} + ... + a_b \times 2^0$. The question then becomes, how can we find those weights $\{a_1, a_2, ..., a_b\}$ that make the sum seven? The answer is division.

Value of Quotient	**Remainder**	**Binary Representation**
7	0	
3	1	1
1	1	11
0	1	111
0	0	111

In the table above, we perform the division necessary to calculate the binary representation of seven in base two. Each time we divide the decimal by two and get a remainder of one, we put a one in the rightmost place of our decimal representation. Likewise, if we have a remainder of zero, we append a zero. Once the value of the quotient is zero, we halt computation because we have found the binary representation in full.

In the first row, we have the original decimal, a remainder of zero since we have not begun to divide yet, and an empty binary representation. The second row has the whole number portion of the quotient, the remainder, and the beginnings of our binary representation. The third and fourth rows continue filling out binary representation. The fifth row is where we meet the halt condition, which is why the binary representation is unchanged.

So know that we know our binary representation is 111_2, we can say that $7_{10} = 1 \times 2^2 + 1 \times 2^1 + 1 \times^0 = 111_2$.

1.2 Decimal and Binary Fractions

Being able to rewrite whole numbers in exponential form begs a question: can we do the same for fractions? The answer is of course, yes.

Given a decimal fraction, like 0.625, we can use negative exponents to write it in exponential form, like so: $0.625 = 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$. How can we express 0.625 as a binary fraction? An intuitive leap to make is that since converting whole numbers to binary involved division, converting fractions might involve multiplication (which happens to be the case).

2 Algorithms

2.1 A Simple Approach

Times 2 table

Value of Product	**Append**	**Subtract from Decimal**
$(0, 1)$	'0'	0×2^0
$[1, 2)$	'1'	1×2^0

2.2 Exploiting Powers of Two

Times 4 table

Value of Product	**Append**	**Subtract from Decimal**
$(0, 1)$	'00'	$0 \times 2^1 + 0 \times 2^0$
$[1, 2)$	'01'	$0 \times 2^1 + 1 \times 2^0$
$[2, 3)$	'10'	$1 \times 2^1 + 0 \times 2^0$
$[3, 4)$	'11'	$1 \times 2^1 + 1 \times 2^0$

Times 8 table

Value of Product	**Append**	**Subtract from Decimal**
$(0, 1)$	'000'	$0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
$[1, 2)$	'001'	$0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$[2, 3)$	'010'	$0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$[3, 4)$	'011'	$0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
$[4, 5)$	'100'	$1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
$[5, 6)$	'101'	$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$[6, 7)$	'110'	$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$[7, 8)$	'111'	$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

Times 16 table

Value of Product	**Append**	**Subtract from Decimal**
$(0,1)$	'0000'	$0\times2^3+0\times2^2+0\times2^1+0\times2^0$
$[1,2)$	'0001'	$0\times2^3+0\times2^2+0\times2^1+1\times2^0$
$[2,3)$	'0010'	$0\times2^3+0\times2^2+1\times2^1+0\times2^0$
$[3,4)$	'0011'	$0\times2^3+0\times2^2+1\times2^1+1\times2^0$
$[4,5)$	'0100'	$0\times2^3+1\times2^2+0\times2^1+0\times2^0$
$[5,6)$	'0101'	$0\times2^3+1\times2^2+0\times2^1+1\times2^0$
$[6,7)$	'0110'	$0\times2^3+1\times2^2+1\times2^1+0\times2^0$
$[7,8)$	'0111'	$0\times2^3+1\times2^2+1\times2^1+1\times2^0$
$[8,9)$	'1000'	$1\times2^3+0\times2^2+0\times2^1+0\times2^0$
$[9,10)$	'1001'	$1\times2^3+0\times2^2+0\times2^1+1\times2^0$
$[10,11)$	'1010'	$1\times2^3+0\times2^2+1\times2^1+0\times2^0$
$[11,12)$	'1011'	$1\times2^3+0\times2^2+1\times2^1+1\times2^0$
$[12,13)$	'1100'	$1\times2^3+1\times2^2+0\times2^1+0\times2^0$
$[13,14)$	'1101'	$1\times2^3+1\times2^2+0\times2^1+1\times2^0$
$[14,15)$	'1110'	$1\times2^3+1\times2^2+1\times2^1+0\times2^0$
$[15,16)$	'1111'	$1\times2^3+1\times2^2+1\times2^1+1\times2^0$