Number Theory

Content

Introduction

Number theory was once viewed as a beautiful but largely useless subject in pure mathematics. Today number-theoretic algorithms are used widely, due in large part to the invention of cryptographic schemes based on large prime numbers. These schemes are feasible because we can find large primes easily, and they are secure because we do not know how to factor the product of large primes (or solve related problems, such as computing discrete logarithms) efficiently.

It is convenient to think of the elementary arithmetic operations (multiplications, divisions, or computing remainders) as primitive operations that take one unit of time. By counting the number of such arithmetic operations that an algorithm performs, we have a basis for making a reasonable estimate of the algorithm's actual running time on a computer. Elementary operations can be time-consuming, however, when their inputs are large. It thus becomes convenient to measure how many bit operations a number-theoretic algorithm requires.

Number System

Number System is a method of representing Numbers on the Number Line with the help of a set of Symbols and rules. These symbols range from 0-9 and are termed as digits. The Number System is used to perform mathematical computations ranging from great scientific calculations to calculations like counting the number of Toys for a Kid or Number chocolates remaining in the box. Number Systems consist of multiple types based on the base value for its digits.

Based on the base value and the number of allowed digits, number systems are of many types. The four common types of Number System are:

1. Decimal Number System - Base = 10
2. Binary Number System - Base = 2
3. Octal Number System - Base = 8
4. Hexadecimal Number System - Base = 16

Binary Representation

Binary is a base-2 number system that uses two states 0 and 1 to represent a number. We can also call it to be a true state and a false state. A binary number is built the same way as we build the normal. For example, a decimal number 45 can be represented as 4*10^1+5*10^0 = 40+5

Now in binary 45 is represented as 101101. As we have powers of 10 in decimal number similarly there are powers of 2 in binary numbers. Hence 45 which is 101101 in binary can be represented as: 2^0*1+2^1*0+2^2*1+2^3*1+2^4*0+2^5*1 = 45. The binary number is traversed from left to right.

Sign and Magnitude representation

There are many ways for representing negative integers. One of the ways is sign-magnitude. This system uses one bit to indicate the sign. Mathematical numbers are generally made up of a sign and a value. The sign indicates whether the number is positive, (+) or negative, (–) while the value indicates the size of the number.

For example 13, +256 or -574. Presenting numbers this way is called sign-magnitude representation since the leftmost digit can be used to indicate the sign and the remaining digits the magnitude or value of the number.

Sign-magnitude notation is the simplest and one of the most common methods of representing positive and negative numbers. Thus negative numbers are obtained simply by changing the sign of the corresponding positive number, for example, +2 and -2, +10 and -10, etc. Similarly adding a 1 to the front of a binary number is negative and a 0 makes it positive.

For example 0101101 represents +45 and 1101101 represents -45 if 6 digits of a binary number are considered and the leftmost digit represents the sign.

But a problem with the sign-magnitude method is that it can result in the possibility of two different bit patterns having the same binary value. For example, +0 and -0 would be 0000 and 1000 respectively as a signed 4-bit binary number. So using this method there can be two representations for zero, a positive zero 0000 and also a negative zero 1000 which can cause big complications for computers and digital systems. The 2's complement notations used to represent signed magnitude numbers are:

Two's complement – Two's Complement is another method like one's complement form, which we can use to represent negative binary numbers in a signed binary number system. In two's complement, the positive numbers are exactly the same as before for unsigned binary numbers. A negative number, however, is represented by a binary number, which when added to its corresponding positive equivalent results in zero.

Bitwise Operators - In C, the following 6 operators are bitwise operators (work at bit-level).

1. The **& (bitwise AND)** in C or C++ takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. The **| (bitwise OR)** in C or C++ takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
3. The **^ (bitwise XOR)** in C or C++ takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. The **<< (left shift)** in C or C++ takes two numbers, left shifts the bits of the first operand, and the second operand decides the number of places to shift.
5. The **>> (right shift)** in C or C++ takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.
6. The **~ (bitwise NOT)** in C or C++ takes one number and inverts all bits of it.