一种快速的随机数生成算法

Alvin 2019/01/23
idealvin@qq.com

这篇文章将介绍一种快速、高效的随机数生成算法，它从一个给定的整数开始，无重复的生成 1 到 $2^{31}-2$ 之间的整数，这些数全部遍历完后，才会进入周期性循环。google 的 leveldb 中有这个算法，有兴趣可以去看看 leveldb 的源码。

基本思想

下面将介绍这种算法的数学原理。首先记:

$$m = 2^{31} - 1 = 2147483647$$

容易验证 m 是一个素数。接下来找一个小于 m 的正整数 a，显然 a 与 m 互素，至于怎么选取 a 后面再详述。现在从一个给定的正整数 $s_0 < m$ 开始，按照下面的方式生成后续的数:

$$\begin{gathered} s_1 \equiv s_0 \cdot a \pmod{m} \\ s_2 \equiv s_1 \cdot a \pmod{m} \\ \vdots \\ s_{m-1} \equiv s_{m-2} \cdot a \pmod{m} \end{gathered}$$

符号 $\equiv$ 表示同余，s_1 是 $s_0 \cdot a$ 除以 m 的余数，其它类推。其中 s_0 不能是 0 或 m，否则后面的数全是 0。现在来证明

$$s_{m-1} = s_0$$

这里需要用到同余的性质以及**费马小定理**，可以参考《费马小定理及其欧拉推广》。由同余的性质，可以得到:

$$s_1 \cdot s_2 \ \cdots \ s_{m-1} \equiv s_0 \cdot s_1 \ \cdots \ s_{m-2} \ \cdot a^{m-1} \pmod{m}$$

又由于 m 是素数，m 与小于它的 $s_1, s_2 \cdots s_{m-2}$ 均互素，因此可以得到:

$$s_{m-1} \equiv s_0 \cdot a^{m-1} \pmod{m}$$

即 m 必定整除 $s_0 \cdot a^{m-1} - s_{m-1}$。另外注意到:

$$s_0 \cdot a^{m-1} - s_{m-1} = s_0 \cdot (a^{m-1} - 1) + (s_0 - s_{m-1})$$

由于 m 与 a 互素，由费马定理知 m 整除 $a^{m-1} - 1$，从而推出 m 整除 $s_0 - s_{m-1}$，这样就证明了 $s_{m-1} = s_0$。

上面的结果表明，这种模运算，最多进行 $m-1$ 次，就会进入周期性循环。s_0 作为种子数，s_1，s_2，s_3 $\cdots$ s_{m-1} 就是后续生成的随机数。如果我们能找到合适的 a，使得这 $m-1$ 个数都不相同(相当于不存在比 $m-1$ 更小的周期)，那么这种伪随机的效果就达到了最佳。

寻找合适的 a 值

上面已经看到，从 s_0 开始，经过 $m-1$ 次模运算后，必定有 $s_{m-1} = s_0$，即 $m-1$ 是它的一个周期。容易看出存在一个最小周期 n，n 必定整除 $m-1$，否则，在 $m-1$ 次模运算后，不可能有 $s_{m-1} = s_0$。另外由

$$\begin{aligned} s_1 &\equiv s_0 \cdot a \pmod{m} \\ s_2 &\equiv s_1 \cdot a \pmod{m} \\ &\vdots \\ s_n &\equiv s_{n-1} \cdot a \pmod{m} \end{aligned}$$

可以得到:

$$s_1 \cdot s_2 \ \cdots \ s_n \equiv s_0 \cdot s_1 \ \cdots \ s_{n-1} \ \cdot a^n \pmod{m}$$

加上 $s_n = s_0$，得到 $a^n \equiv 1 \pmod{m}$，由此可知最小周期 n 必定满足如下两式:

$$\begin{aligned} m &\equiv 1 \pmod{n} \\ a^n &\equiv 1 \pmod{m} \end{aligned}$$

反过来，对于给定的整数 a，如果对 $m-1$ 的任意因子 n $(n < m-1)$，$a^n - 1$ 都不能被 m 整除，我们就能断定不存在比 $m-1$ 更小的周期，从而找到所需要的 a 值。这个验证的过程，很容易用计算机程序实现，恐繁不述。

另外需要指出的是，对于 $n < m-1$，若满足 $a^n \equiv 1 \pmod{m}$，则 n 必定是一个周期(即有 $s_n = s_0$)。证明过程与上面证明 $s_{m-1} = s_0$ 是类似的。

下面来看看寻找 a 值的思路。由于模运算中需要计算与 a 的乘积，考虑令 $a = 2^k$，这样乘法运算就转化为位移运算，速度更快，但这样的 a 不满足条件。为什么呢？将 $m-1$ 因式

分解得：

$$m-1=2^{31}-2=2\cdot 3\cdot 3\cdot 7\cdot 11\cdot 31\cdot 151\cdot 331$$

因为 31 是 $m-1$ 的一个因子，令 $n=31$，得到：

$$a^n-1=2^{31k}-1$$

可以看出，它总能被 $m=2^{31}-1$ 整除，表明 31 是它的一个周期。这样只能折衷一下，令 $a=2^k+1$，它只有两个 bit 位为 1，乘法运算仍比其他的数快。另外 a 不能太小，否则后续生成的数中可能会出现一长串 a 的倍数。特别的，k 至少要大于 10，以保证 $a^3>m$。幸运的是，通过简单的测试，没用多久就发现 $k=14$ 时满足条件，这样就找到了合适的 a：

$$a=2^{14}+1=16385$$

在 leveldb 中，$a=16807$，它有 7 个 bit 位为 1。理论上 16385 计算乘法时速度更快，效率方面应该更胜一筹。

Random 类的 C++ 实现

这里给出一段简短的 C++ 代码，与 leveldb 的代码大同小异，只是将 16807 换成了 16385。

```
class Random {
  public:
    explicit Random(unsigned int seed = 1) : _seed(seed & 0x7fffffffu) {
        if (_seed == 0 || _seed == 2147483647L) _seed = 1;
    }

    unsigned int next() {
        static const unsigned int M = 2147483647L;  // 2^31-1
        static const unsigned long long A = 16385;  // 2^14+1

        unsigned long long product = _seed * A;

        _seed = static_cast<unsigned int>((product >> 31) + (product & M));
        if (_seed > M) _seed -= M;

        return _seed;
    }

  private:
    unsigned int _seed;
};
```