防止运算溢出的一些技巧

2017, Jul 1 by Tesla Ice Zhang

我曾经经常写一些把普通运算优化为位运算的代码，这个过程在网上被批判为**没有用**。其实我觉得这非常有用，因为它给我带来了 我的代码跑的很快 的错觉，让我在写代码的时候保持一个良好的心态。但是这次，我将面临全新的挑战。

在写一些 JavaScript 代码的时候（其实是把一份 C++ 代码翻译成 JavaScript），我需要处理 JavaScript 由于 Number 仅支持到 2^{53}（这蹋玛是哪来的神奇数字）（其实是浮点实现的锅），在进行一些需要比较大精度的运算时就会出现一些诡异的问题。

为什么我需要把位运算优化成普通运算

- 因为位运算仅支持到 32 位，因此我 53 位左右的数据就会被卡掉。

我们来看一段魔幻现实主义的 REPL 结果：

```
> 28399128732189371
< 28399128732189372
> 28399128732189371 << 1
< -670661256
```

```
> 28399128732189371 >> 1
< 906076510
```

来服不服

This is JavaScript

我还需要做什么

- 原本的快速幂系列肯定不够用了
- 原本有快速幂和快速乘
- 我现在需要快速加

第零关：乘法溢出

其实就是加个快速乘，下面会一起讲。 Voile 作为出题人就没考虑这个情况，我考虑了，然后他的标程就是错的，我的是对的，然后 error 了半天。哼。

第一关：逆优化位运算

首先我们来看一段朴素的快速乘：

```
fastMul = (a, b, m) => {
  var ret = 0;
  a %= m, b %= m;
  while (b) {
    if(b & 1) ret = (ret + a) % m;
    b >>= 2, a = (a << 1) % m;
  }
```

```
  return ret;
}
```

科普一下

快速乘就是用于计算 $(a \times b)\%m$ 的一个巧办法，可以避免 $a \times b$ 过大导致整数溢出而使结果不准确。
这个快速乘看起来和 C++ 实现一毛一样，唯一的区别就是函数和变量声明的语法不一样了。然后我们开开心心地在程序里面使用了这个快速乘。

坑

然后我们发现我们的输出里面出现了负数。 TAT
这其实就是因为在进行 $(32, 53]$ 位的位移运算时， JavaScript 擅自把前面几位给掐掉了（如文章开头那段 REPL 记录所示）。因为这垃圾的位运算仅支持 32 位嘛。
但是由于掐掉前面的位并不影响 $n\&1$ 这个操作，因此我们还能继续使用 `&` 。

This is JavaScript

于是我们就需要把其中的位运算转换成普通的加减乘除运算：

```
fastMul = (a, b, m) => {
  var ret = 0;
  a %= m, b %= m;
  while (b) {
    if(b & 1) ret = (ret + a) % m;
    b = Math.floor(b / 2), a = (a + a) % m;
//      ^^^^^^^^^^^^^^^^^^      ^^^^^^^
```

```
  }
  return ret;
}
```

好，这下看起来稳如狗了。
然后当时的代码还有另外一个地方，有一句

```
var width = 1 << m;
```

这种东西需要被换成：

```
var width = Math.pow(2, m);
```

嗯，这下是真的啥都没有了。
你以为这样就万事大吉了吗？ Naive 。
然后我们发现我们的输出里面出现了负数 again 。 TAT

第二关：危险的加法

这里的原因实际上是因为加法溢出。大概是这样：

```
var a = 9007199254740992;
var mod = 1000000007;
(a + a) % mod;
```

这几个数都是合法的 JavaScript 的 `Number` ，但这个加法会导致溢出整数。一个比较常见的解决方案是：

```
var a = 9007199254740992 - 1;
//                       ^^^^^
var mod = 1000000007;
```

```
(a % mod + a % mod) % mod;
//^^^^^^     ^^^^^
```

这样就不会溢出了。然而，当 mod 等于 a + 1 之类的比较危险的数字之后，这个取模的方法就无效了。
所以说我们需要引入一个新的函数叫 快速加，用来避免以上悲剧发生（至于这个实现是什么意思，希望读者自己领会，我觉得它非常容易读懂）：

```
fastPlus = (a, b, m) => {
 var s = a + b;
 if (s >= a && s >= b) return s % m;
 if (a > m) return plus(a % m, b, m);
 if (b > m) return plus(a, b % m, m);
 return a - m + b;
}
```

然后把所有的加法换掉：

```
fastMul = (a, b, m) => {
  var ret = 0;
  a %= m, b %= m;
  while (b) {
    if(b & 1) ret = fastPlus(ret, a, m);
    b = Math.floor(b / 2), a = fastPlus(a, a, m);
//                              ^^^^^^^
  }
```

```
    return ret;
}
```

现在就万事大吉了对吗？

似乎是的。。。因为这样处理的话只要输入数据是合法的整数，那这一套理论就可以保证正常的工作。

Tweet this

Top

创建一个 issue 以申请评论

Create an issue to apply for commentary

协议/License