LeetCode/068. Sqrt(x) at master · pezy/LeetCode

https://github.com/pezy/LeetCode/tree/master/068. Sqrt(x)

None Tue Sep, 07 01:12

很显然，这又是一道轮子题。有时候我们经常使用某个库函数，的确会偶尔推测它底层是如何实现的。譬如这个 `sqrt`，它如何做到快准狠？

很好奇的看了一下源码，发现 gcc 底层调用了一个 buildin 版本的 sqrtf, 然后就无疾而终了。网上搜一下，大家都只是再说雷神里那惊天地泣鬼神的魔数算法。实际上，就现在的机器性能而言，这个方法并不一定是最合适的。有兴趣者可以参考以下两篇论述：

1. 游戏中的优化指的是什么 - Milo Yip的回答
2. Best Square Root Method - Algorithm - Function (Precision VS Speed)

可以发现，真的想要极致的速度，用 ASM 中的内置函数才是较为明智的选择。

抛开上述轶事不谈，平心而论此题如果只是一道寻常数学题，用程序应该如何解之。

首先最自然的方案，当然是对对碰。如求 10 的平方根，我可以从 1 开始碰，`1*1==1` 显然不对，一直到 `4*4==16` 才发现超出了。于是答案显然是 3.

但这种类似穷举的方式显然很不智，既然是查找能碰对的数，肯定二分搜索要快一些。如先看 `5*5==25`, 25 > 10, 范围向左缩小，并继续中分，有 `3*3==9`，9 < 10，范围向右缩小，并继续中分，有 `4*4==16`, 16 > 10, right == 3, left == 4. 结束。

那结果应该取哪一次的 middle 呢？显然因为咱们是 integer 的运算，取小不取大，3 可以是结果，4 决计不是。故有：

```
if (mid <= x / mid) ret = mid;
```

整个程序非常简单，而且高效 AC:

```
int l = 1, r = x, ret = 0;
while (l <= r) {
    int m = (l + r) >> 1;
    if (m <= x / m) { l = m+1; ret = m; }
    else r = m-1;
}
return ret;
```

这道题算是告一段落，但我们其实占了 integer 的便宜，假使要实现的是 `float sqrtf(float x)`，我们可能需要考虑一下使用著名的牛顿迭代了。这就基本演变为一道数学题了。具体可见 Matrix67 这篇博文中的解释。

```
#include <cfloat>
#include <cmath>

float sqrtf(float x) {
    float ret;
    for (float f = 1.f; true; f = ret) {
        ret = (f + x / f) / 2;
        if (std::abs(f - ret) < FLT_MIN) break;
    }
    return ret;
}
```