非連続な実装を持つ連続値の境界値分析の考察

秋山 浩一
富士ゼロックス株式会社
Kouichi.Akiyama@fujixerox.co.jp

要旨

2016 年の閏日(20160229)は素数か否か？ という素朴な疑問から「ある年に素数はいくつあるか」という問題に発展し，この問題を解くプログラムが生まれた．ところがそのプログラムには境界値にまつわるバグがあった．本稿はそのバグの分析と対応について考察したものである．

年月日を“YYYYMMDD”の 8 ケタの数値で表現した場合その数値は必ず増加するが，月変わりや年変わりの境界では非連続となる．今回のバグは年変わりの境界で発生した．プログラムは容易にデバッグできたが，デバッグ後のプログラムが正しく動作することの検証について，計算量が多く答えがでなかった．冷静になって考え直したところ，数学的に解ける問題であることがわかった．

この経験を公開することで境界値にまつわるやっかいなバグの解決ヒントになることを期待している．

1. 背景

「閏日である2016年2月29日は素数か？」という問題が発せられた．年月日を「YYYYMMDD」の 8 ケタの数値で表現したときに「20160229」は素数か否かという問題である．素因数分解の簡単なプログラムを作成することですぐに「20160229＝929×21701」という解が得られるので「20160229 は素数ではない」という答えが得られた．

ここで，筆者は「年月日を同様の 8 ケタの整数で表現したときに2016年に素数はいくつ存在するか」という問題を考えてみた．簡単なプログラムを書き「素数は 19 個で内訳は 20160319, 20160401, 20160403, 20160529, 20160601, 20160607, 20160611, 20160709, 20160727, 20160809, 20160817, 20160821, 20160923, 20161007, 20161013, 20161019, 20161021, 20161027, 20161103 に違いない」という内容のツイートをしたところ知人でプログラムが趣味の居駒氏から，

```
ruby -rdate -rprime –e
'p    (Date.new(2016,1,1)..Date.new(2016,12,31))
.select{|date|date.strftime("%Y%2m%2d").to_i.prime?}.size'
```

という 1 ラインコードを DM で受け取った．その後，居駒氏から「上記は 140 文字に収まらないので」と，

```
ruby -rdate -rprime -e
'p (0..366).count{|i|(http://Date.new(2016,1,1)+i)
.strftime("%Y%m%d").to_i.prime?}'
```

という別解のツイートがあったが，改変後のプログラムには境界値にまつわるバグがあった．

2. 考察と解決策

2.1. バグの内容とデバッグ

バグはごく単純なもので「(0..366)」では 367 日チェックしてしまうという内容であった．「20170101」は素数ではないため，この境界値を含んでも含まなくても結果は 19 で改変前のプログラムの出力結果と変わらないという落とし穴もあった．

デバッグは「.」を一つ増やし「(0...366)」とすることで366を含まなくすればよい．

このとき，366を365に変えても同様の結果が得られるが，それではプログラミングの可読性(意図の獲得性)が悪くなるので，本ケースにおいては，Ruby の範囲式のうち終端を含まない「...」を使用すべきである．

2.2. デバッグ結果の検証

さて，デバッグ後の回帰テストについて居駒氏は「当該年の 1/1 と 12/31 に加えて，前年の 12/31 と翌年の 1/1 の 4 つのすべてが素数である年をテストする必要がある」と考えた．そこで，プログラムを組み，そのような年が存在するかのチェックを行った．それが見つかれば，その年で回帰テストを行えばよいからである．

ところが，2016 万をはるかに越えた 5 億年まで計算しても 3 つまで素数の年は数多くあるが 4 つ全てが素数の年は見つからない．その後，丸 1 日プログラムを走らせ続けても見つからなかった．(このときの検証環境は

OpenMP で 8 コア並列であった）

2.3. 問題の解決と考察

さて，プログラムにより4つの数字全てが素数の年がなかなか見つからないため，そもそもそのような年は無いのではないかと考えた．

存在することをプログラムで示すことはできるが，存在しないことをプログラムで証明することは難しい．たとえ，1兆年まで確認しても1兆1年で成立するかもしれないからである．（いわゆる「悪魔の証明」である）

そこで，プログラムではなく数学での証明を試みた．今回テストしたい数値は，前年の 12/31，当該年の 1/1 と 12/31，翌年の 1/1 の4つである．2016 年でいえば {20151231, 20160101, 20161231, 20170101}の 4 つの数字の全てが素数であるパターンである．

「これらの 4 つの数字が全て素数となることはない」ことについて，居駒氏により数学的に証明できることが分かった．

まず，この 4 つの数字を

（x1231，y0101，y1231，z0101 where x+1 = y = z-1）

と定義する．

素数は，5 以上では，6 で割った時の余りが1か5しか候補にならない．したがって x1231 が素数の場合，その余りは1か5である．

ここで，y0101-x1231 は，常に 8870 である．8870 を 6 で割った時の余りは 2 であるから，x1231 が余り 5 で，y0101 が余り 1 でないと両方素数にならない．

y0101 と y1231 の差は，1230 で，1230 を 6 で割った時の余りは 2 である．

したがって，x1231 と y0101 の両方が素数のとき，y1231 は必ず 6 で割ると 3 余りとなり，これは素数では無い．したがって，4 つの数字が全て素数であることはあり得ない．

さて，この経験からの学びをまとめよう．筆者は 3 つの学びがあったと考えている．一つ目は，日付が連続的に増えていくのに対して，年月日を“YYYYMMDD”の 8 ケタの数値で表現した場合の数値は非連続で増加するために実装上の境界が生じていたことである．このように仕様上あるいは日常的な感覚で連続性があっても実装上は飛び地があり，そこが境界になっている場合がある．ソフトウェアテストにおいてはまず，このような内部的に発生している境界に気がつく必要がある．そこに欠陥が生じやすいためである．

次に，閏年と言うことで 366 という値をプログラムに書き，範囲式を誤るという欠陥があった．これは，一つ注意したことで気持ちの中の警戒心が緩んだのであろう．「閏年についての考慮はしてある」という安心感が欠陥を誘発したのである．一つの落とし穴に対してプログラム上では複数の対応が必要な事がある．間違えやすいポイントは何度もチェックする必要があろう．

最後の学びは，一つの解決手段にこだわり過ぎると失敗するということである．数学的に証明しようと思えば 1 時間もかからずに済む問題が，プログラミングで答えを求めようとこだわると結果が得られなかった．

本件はシンプルな事例であるが，商品におけるやっかいなバグや，その解決方法の選択ミスで似たような経験を何度もしている．この事例から得た教訓を実業務で活かしていくことが重要であると考える．