06-22-2018, 10:20 PM **Post: #41**

Jeff O.
Member

Posts: 166
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

With some coaching and insights from Valentin, I attacked the problem from a more efficient direction and managed to create a program which is at least three orders of magnitude faster than my previous brute-force efforts. The programs below calculate the 37 seflies from 1 to 11 digits in approximately 1 minute, 6 seconds on my probably not too fast desktop. Extrapolating run times, I calculated that it could run in something less than 3 hours on my DM42. So I plugged it into a USB power source and set it off. Two hours, 34 minutes later it completed the task, so it apparently is a realistic challenge for a physical machine.

The most useful information that Valentin provided was that it is not necessary to check and sum every number. While there are 89,999,999,999 11 digit numbers from 10,000,000,000 to 99,999,999,999, many, many, many of these contain the same digits and so will have the same sums to the 11th power. So if a way can be found to sum each combination just once, that will greatly reduce the quantity of numbers needing to be checked. (e.g., 54321 is the same as 43215 is the same as 12345 etc., so only one of these need be summed.) I think I actually realized this while working on the brute-force methods, but saw no easy way to generate only the minimal set of combinations of digits. With Valentin's encouragement that this was the way to go, I was able to develop a method to sequentially generate the smallest combination of the digits for each set of n digits. The method may be difficult to decipher from the program listing (it happens in steps 10 through 42). I doubt that the method is particularly inisghtful, so in an effort to keep this post as short as possible, I won't describe it. I can do so if there is any interest.

Once the sum for a given set of n digits is created, you just have to see if its sum of digits to the nth power contains the same set of digits

as the input number. If so, the reverse of that sum is a selfie. I did this in what I am sure is a very clunky fashion:

- Break the number into its separate digits
- Sort the digits
- Reassemble into a number
- Compare to input number
- if equal, reverse the sum, that is the selfie
- if not equal, not a selfie, go back to create the next combination to check

Due to the way I "manufacture" the numbers to check, having to do with how I handled numbers with zeros in them, my program generates the selfies in the following order:

1
2
3
4
5
6
7
8
9
704
351
173
8028
4361
48039
4749
7180089
72729
84745
8180124
438845
5271471
5136299
15087642
802115641
77439588
351589219

52249382004
30609287624
579533274
638494435
4777039764
60605588394
15694046123
41919540249
97653680744
87561939628

The program is far from optimized, it would probably be possible to improve on the performance. But I believe that Valentin plans to wrap this one up soon, and I'm not sure I will have the time to try for further improvement. I wanted to get an "entry" in, so I'll go ahead and post this version. If I do find time and am able to improve the program before Valentin finalizes things, I'll provide an update.

```
00 { 249-Byte Prgm }
01▸LBL "NM5"
02 11
03 STO 00
04▸LBL 01
05 0
06 STO IND 00
07 DSE 00
08 GTO 01
09▸LBL 02
10 1.011
11 STO 00
12▸LBL 03
13 9
14 RCL IND 00
15 X≠Y?
16 GTO 04
17 ISG 00
18 GTO 03
19 STOP
20▸LBL 04
21 RCL 00
22 IP
23 STO 00
```

```
24 1
25 RCL+ IND 00
26▸LBL 05
27 STO IND 00
28 DSE 00
29 GTO 05
30 11
31 STO 00
32 0
33▸LBL 06
34 RCL 00
35 1
36 -
37 10↑X
38 RCL× IND 00
39 +
40 DSE 00
41 GTO 06
42 STO 12
43 CLA
44 CF 29
45 FIX 00
46 ARCL ST X
47 ALENG
48 STO 00
49▸LBL 09
50 0
51▸LBL 07
52 ATOX
53 X=0?
54 GTO 08
55 48
56 -
57 RCL 00
58 Y↑X
59 +
60 GTO 07
61▸LBL 08
62 R↓
```

```
63 STO 14
64 LOG
65 IP
66 1
67 +
68 RCL 00
69 X≠Y?
70 GTO 10
71 RCL 14
72 10
73 ÷
74 FP
75 X=0?
76 GTO 10
77 ARCL 14
78 RCL 00
79 20
80 +
81 1000
82 ÷
83 21
84 +
85 STO 16
86 STO 15
87▸LBL 11
88 ATOX
89 X=0?
90 GTO 13
91 48
92 -
93▸LBL 13
94 STO IND 15
95 ISG 15
96 GTO 11
97 ISG 13
98 DEG
99 XEQ "SORT"
100 20.02
101 RCL+ 00
```

```
102 STO 15
103 0
104▸LBL 12
105 RCL 15
106 21
107 -
108 IP
109 10↑X
110 RCL× IND 15
111 +
112 DSE 15
113 GTO 12
114 RCL 12
115 X≠Y?
116 GTO 10
117 RCL 14
118 STO- 14
119▸LBL 14
120 FP
121 STO+ 14
122 LASTX
123 IP
124 0.1
125 STO÷ 14
126 ×
127 X=0?
128 GTO 15
129 GTO 14
130▸LBL 15
131 RCL 14
132 PRX
133▸LBL 10
134 11
135 RCL 00
136 X=Y?
137 GTO 02
138 RCL 12
139 ARCL ST X
140 ISG 00
```

```
141 DEG
142 GTO 09
143 STOP
144 END

00 { 51-Byte Prgm }
01▸LBL "SORT"
02 RCL 16
03 SIGN
04▸LBL 21
05 LASTX
06 LASTX
07 RCL IND ST L
08▸LBL 22
09 RCL IND ST Y
10 X<Y?
11 GTO 23
12 X<>Y
13 LASTX
14 +
15▸LBL 23
16 R↓
17 ISG ST Y
18 GTO 22
19 X<> IND ST L
20 STO IND ST Z
21 ISG ST L
22 GTO 21
23 RTN
24 END
```

Credit to Gamo for the reversing integer routine in steps 117 through 129.
Credit to Jean-Marc Baillard for the SORT subroutine.

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

06-24-2018, 04:57 PM **Post: #42**

Thomas Klemm
Senior Member
Posts: 1,449
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Here's a Python program for the sixth step:

Code:

```
def selfie(m, i, k, n, s):
    if i == 0:
        if n == ''.join(sorted(str(s))):
            print str(s)[::-1]
    else:
        for j in range(k, 10):
            selfie(m, i - 1, j, n + str(j), s + j ** m)

for m in range(12):
    selfie(m, m, 0, '', 0)
```

The following 41 selfies are listed:

Code:

```
0
1
2
3
4
5
6
7
8
9
073
```

It takes about 3 seconds to run:

Code:

```
real    0m2.805s
user    0m1.942s
sys     0m0.054s
```

Four of them start with 0:

Code:

```
0
073
05087642
05694046123
```

They might not be considered selfies which agrees with:

Quote:

all 37 Selfies up to 11 digits long

PM FIND QUOTE REPORT

06-25-2018, 02:42 PM **Post: #43**

John Keith
Senior Member

Posts: 389
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Thomas Klemm Wrote: ➡ (06-24-2018 04:57 PM)

Here's a Python program for the sixth step:

Code:

```
def selfie(m, i, k, n, s):
    if i == 0:
        if n == ''.join(sorted(str(s))):
            print str(s)[::-1]
    else:
        for j in range(k, 10):
            selfie(m, i - 1, j, n + str(j), s + j ** m)

for m in range(12):
    selfie(m, m, 0, '', 0)
```

I gather that Valentin's original definition of a selfie as the *reverse* of the sum of powers is aimed at eliminating numbers which end in zero. I'm not a Python expert, but if you amend the third line of your program to test that the last digit of s is not zero, it will return Valentin's

original 37 numbers (at some cost in execution speed).

John

EMAIL PM FIND QUOTE REPORT

06-25-2018, 03:41 PM **Post: #44**

Thomas Klemm
Senior Member

Posts: 1,449
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **John Keith Wrote:** ⇨ (06-25-2018 02:42 PM)
> I gather that Valentin's original definition of a selfie as the *reverse* of the sum of powers is aimed at eliminating numbers which end in zero.

Not sure about that. Might as well be to make it a bit more difficult.

> **Quote:**
> I'm not a Python expert, but if you amend the third line of your program to test that the last digit of s is not zero, it will return Valentin's original 37 numbers (at some cost in execution speed).

Or then you simply filter them out after the computation:

Code:

```
python selfie.py | grep -v ^0
```

PM FIND QUOTE REPORT

07-18-2018, 02:17 AM **Post: #45**

Valentin Albillo
Senior Member

Posts: 347
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.

Hi, all:

This is my **200th** post so it's perfectly fitting to wrap up here and now my ***S&SMC#23*** by posting my very own solution to the final *6th Step*, namely:

Step the Sixth:

- *"We'll call "Selfie" to any positive N-digit integer number which has the property that if you sum its N digits raised to the Nth power you get the original number backwards. For instance, the 7-digit number 5271471 is a Selfie:*

  5271471 => 5^7 + 2^7 + 7^7 + 1^7 + 4^7 + 7^7 + 1^7 = 1741725, which is 5271471 backwards

  Write a program to find all Selfies from 1 to 9 digits long (for 10-digit HP calcs, 29 in all) or from 1 to 11 digits long (for 12-digit HP calcs, 37 in all). 0 is not a positive number so it's not a Selfie."

My Solution:

This is my solution for the **HP-71B**, a 12-line program (437 bytes) which finds and outputs all *Selfies* from 1 up to 11 digits long in less than 3 min on my *POPS* system.:

```
 1  DESTROY ALL @ OPTION BASE 0 @ DIM R(9) @ L=48 @ FOR K=1 TO 11
 2  DIM F(K),S(K),D$(K) @ DISP K;": "; @ M=10^K @ T=M/10
 3  H=0 @ FOR I=1 TO 9 @ R(I)=I^K @ NEXT I
 4  H=H+1 @ F(H)=F(H-1)
 5  I=F(H) @ N=S(H-1)+R(I) @ IF N<M THEN S(H)=N @ D$(H)=D$(H-1)&CHR$(I+L) ELSE 11
 6  IF H#K THEN 4 ELSE IF N<T THEN 10 ELSE B$=STR$(N) @ A$=D$(H)
 7  IF SPAN(B$,A$) THEN 10 ELSE IF SPAN(A$,B$) THEN 10
 8  FOR I=1 TO K @ P=POS(A$,B$[I,I]) @ IF P THEN A$[P,P]="" ELSE 10
 9  NEXT I @ B$=REV$(B$) @ IF B$=TRIM$(B$,"0") THEN DISP B$;" ";
10  IF F(H)#9 THEN F(H)=F(H)+1 @ GOTO 5
11  H=H-1 @ IF H THEN 10
12  DISP @ NEXT K
```

Notes:

- We only search for numbers having from 1 to 11 digits because for 12-digit numbers there are some whose sum of the 12th-powers of their digits exceeds 10^12 and thus cannot be checked correctly with 12-digit integer arithmetic. For instance:

  888888888888 -> Sum = 12*8^12 = 824633720832, which is still within range and can be correctly checked
  888888888889 -> Sum = 11*8^12+1*9^12 = 1.03834378058E12, which *exceeds* the integer range and *can't* be checked properly

thus the search is limited to 11-digit numbers or less (there are no 12-digit Selfies anyway). The same applies to 10-digit calcs, which can only search for numbers up to 9-digit long.
- *Line 7* uses the *SPAN* keyword from *STRNGLEX* to help speed the search but it's use is optional and can simply be deleted. Without it the program is *8% shorter* (11 lines, 371 bytes) but *18% slower* (200 sec. vs. 170 sec.)

Let's run it:

```
>RUN
         1 : 1 2 3 4 5 6 7 8 9
         2 :
         3 : 704 351 173
         4 : 8028 4361 4749
         5 : 48039 72729 84745
         6 : 438845
         7 : 7180089 8180124 5271471 5136299
         8 : 15087642 77439588
         9 : 802115641 351589219 579533274 638494435
        10 : 4777039764
        11 : 52249382004 30609287624 60605588394 15694046123 41919540249 97653680744 87561939628
```

There are ***37*** *Selfies* up to 11 digits long in all (for 12-digit calcs) and ***29*** *up to 9 digits long* (for 10-digit calcs). There are no 12-digit *Selfies*.

The program uses my *generalized loops* (first featured in ***S&SMC#21***) to perform an exhaustive search for *Selfies*. The key to speed the search enormously is to check as few numbers as necessary (this will reduce the search time *exponentially*), then to implement minor but welcome optimizations which will further reduce the seach time by a significant *linear* factor.

The procedure goes like this: for N-digits numbers, there's no need to check every number from *00...00* to *99...99*, we only need to check the *smallest* N for each permutation of its N-digits. This alone reduces the search exponentially, as stated. Let's see an example with 2-digit numbers:

For 2-digit numbers, in a naive exhaustive search we would simply compute the sum of the 2nd power (squares) of *every* number from *00* to *99* and see if the sum has the *Selfie* property, i.e., it equals the original number in reverse. If so, it is displayed as a solution and we would then proceed to the next 2-digit number and so on. We would check *10^2 = 100* numbers in all, the 100 values from *00* to *99*.

However, we might notice that *12* and *21* have the exact same sum of their squared digits because addition is conmutative: ***Sum(12)*** = *1^2+2^2* ***= 5*** *= 2^2+1^2* ***= Sum(21)***, so we don't actually need to compute and check the sums for every digit permutation, checking just the first one *(12)* will do, no need to also check *21*.

This, combined with the fact that the sum must be 2-digit too (so we need to check only those numbers which have 2-digit sums, i.e.: *10*

>= *Sum(N)* <= *99*) means that instead of the ***100*** numbers from *00* to *99* we only need to check these ***40***:

```
04 05 06 07 08 09 13 14 15 16 17 18 19 23 24 25 26 27 28 29
33 34 35 36 37 38 39 44 45 46 47 48 49 55 56 57 58 66 67 77
```

and thus the search has been reduced from *100* to just *40* numbers, i.e.: by a factor of *2.5x*. For greater number of digits the reduction factor increases *exponentially* and thus the time for the full search decreases exponentially as well.

What do we need to do to implement this concept ? Two things:

- **first**, we need to generate and check only the *lowest* value for every permutation of digits, i.e: for N = 2 digits we'll generate and check *17* but not *71*. For N=3 digits, we'll generate and check *047* but not its five permutations *(074, 407, 470, 704, 740)*, as they all have the same sum of the 3rd powers of their digits. This means that for N=10 digits, you'll only generate and check **one** of the *3,628,800 permutations* possible, thus speeding up the search by a factor of *~ 3.6 million*.
- **second**, for each generated number (say *047*) we have to check if the sum of the Nth powers of its digits is a permutation of the number being checked, i.e.; the sum of the cubed digits of ***047*** is *0^3+4^3+7^3 =* ***407***, which indeed *is* a permutation of *047*, the number being checked, so the reverse of the sum, ***704***, is a *Selfie* and we display it as one of the 3-digit *Selfies*.

  This will reduce the search time exponentially. Combined with other optimizations (generating efficiently the numbers to check, computing efficiently the sums, skipping N-digit numbers with sums > or < N-digits, checking efficiently if an N-digit number is or not a permutation of another, etc.) will help to further reduce the time by a *linear* factor which can be *~ 10x* or more, i.e., further reducing an already fast 30 min. search to *less than 3 min*.

This is not the only efficient approach possible. There's another way to conduct the search by keeping *tallies* but my **HP-71B** implementation of that second approach isn't included here as it runs *slower* than the first (though for some non-*HP-71B* implementations it can run more than 2x faster).

Thanks for your interest and nice solutions, see you all in ***S&SMC#24*** next Autumn.
V.
.

PM FIND EDIT QUOTE REPORT

07-18-2018, 03:10 AM **Post: #46**

rprosperi
Senior Member

Posts: 3,278
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Valentin Albillo Wrote: ➡ (07-18-2018 02:17 AM)

This is my **200th** post so it's perfectly fitting to wrap up here and now my ***S&SMC#23*** by posting my very own solution to the final *6th Step*, namely...

Thanks for the original challenge but also for the several earlier, and now this detailed solution for the final section. Though most often I am not up these challenges of yours, I thoroughly enjoy pondering them (until I get frustrated) and then reading the various members' solutions. And your detailed explanations (like this one above) are excellent learning tools as they not only explain how you solved it, but also why you chose that technique and how it works, generally in a clear enough manner to see how to apply similar techniques in the future to other problems.

Also, I've noted that your very detailed solutions have often inspired others to also explain their solutions in a similar, detailed manner, which is far more helpful to subsequent readers than a (possibly well thought-out, but still) obscure code listing and a brief note claiming this solution is '3 bytes shorter' or some other similar claim to fame.

Thanks to all that participated and shared their answers here, it's quite interesting to many of us quiet readers.

--Bob Prosperi