HP Forum Archive 21

[Return to Index | Top of Index]

Weekend programming challenge: Euler's Totient function

*Message #1 Posted by **Allen** on 1 June 2012, 10:55 p.m.*

Greetings fellow calculator enthusiasts! If you are up for a nice weekend challenge, I would like to propose a small challenge that has some very interesting properties and applications.

Your challenge is to write a small program, which given any number "a" will calculate phi(a) (Euler's Totient series) where, phi(a) is the number of relatively prime numbers less than or equal to a.

There are several ways to accomplish this- perhaps you have some creative approaches we can share and learn??

Re: Weekend programming challenge: Euler's Totient function

*Message #2 Posted by **Valentin Albillo** on 2 June 2012, 2:15 a.m.,*
in response to message #1 by Allen

> Quote:
>
> Your challenge is to write a small program, which given any number "a" will calculate phi(a) (Euler's Totient series) where, phi(a) is the number of relatively prime numbers less than or equal to a.
>
> There are several ways to accomplish this- perhaps you have some creative approaches we can share and learn??

Easy. Just take an **HP-71B** fitted with the **JPC ROM** (both available for free in their emulated form) and there you are:

```
10 INPUT N @ DISP PHI(N)

>RUN
  ? 45
         24

>RUN
  ? 46
         22

>RUN
```

```
? 98765430111
               61497371520
```

... in negligible time. Come to think of it, you don't need the 1-line program, just use PHI(98765430111), etc., right from the command line ... XD

Regards from V.

Re: Weekend programming challenge: Euler's Totient function

*Message #3 Posted by **Gilles Carpentier** on 2 June 2012, 3:06 a.m.,*
in response to message #2 by Valentin Albillo

With a standard 50G

45 EULER
24

;)

If "Euler" does not exist, a first version not optimised at all :

```
«
 0 OVER
 1 SWAP FOR n
  OVER n GCD 1 ==  +
 NEXT
 NIP
»
```

Edited: 2 June 2012, 3:36 a.m.

Re: Weekend programming challenge: Euler's Totient function

*Message #4 Posted by **Ángel Martin** on 2 June 2012, 3:42 a.m.,*
in response to message #3 by Gilles Carpentier

Maybe I didn´t read the original posting well, but I think the challenge was about programming the PHI function - not about executing the function from machines that have it implemented.

TOTNT is also available in my ¨ALGEBRA¨ module for the 41, and in the i41CX emulator. I used a brute-force approach that requires the prime factors of the argument, so nothing to write home about.

An article is available here:

http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/articles.cgi?read=928

Re: Weekend programming challenge: Euler's Totient function

*Message #5 Posted by **Gilles Carpentier** on 2 June 2012, 3:50 a.m.,*
in response to message #4 by Ángel Martin

Yes ! But i must say it took me some time to see that the PHI function was natively implemented in the HP50, so this can help...

I have edited my post with a very trivial and inefficient RPL version.

Edited: 2 June 2012, 3:51 a.m.

Re: Weekend programming challenge: Euler's Totient function

*Message #6 Posted by **Gerson W. Barbosa** on 2 June 2012, 5:37 a.m.,*
in response to message #5 by Gilles Carpentier

The following appears to work for all **n** and solves Valentin's example in negligible time on the real HP 50g. I didn't know this function was built into the HP 50g either.

```
%%HP: T(3)A(R)F(,);
%%HP: T(3)A(R)F(,);
\<< DUP
  IF 1 \=/
  THEN { } SWAP FACTORS DUP SIZE 1 SWAP
    FOR i DUP i GET ROT + SWAP 2
    STEP { } SWAP DUP SIZE 2 SWAP
    FOR i DUP i GET ROT + SWAP 2
    STEP DROP 1 - OVER SWAP
    \<< ^
    \>> DOLIST SWAP 1 - + \PILIST
  END
\>>


<< 98765430111 PHI >> TEVAL   -->  61497371520
                                      s: .6544
```

Gerson.

P.S.: This is what I have observed:

```
n = 826875 = 7^2 * 5^4 * 3^3


{ 7   2   5   4   3   3 }     (list of the prime factors of n, followed by their multiplicities)
   \       \       \
    \       \       \
{ 6   1   4   3   2   2 }     (one unit subtracted of all elements)
  -       -       -
```

```
phi(n) = 7^1 * 5^3 * 3^2 * 6 * 4 * 2 = 37800
```

Edited: 2 June 2012, 6:19 a.m.

Re: Weekend programming challenge: Euler's Totient function

*Message #7 Posted by **Gilles Carpentier** on 2 June 2012, 7:36 a.m.,*
in response to message #6 by Gerson W. Barbosa

Hi Gerson ! I dont understand why it works, but it works fine

Here are 4 variations with your algorithm :

```
«
 FACTORS
 2. « NSUB 2. MOD { OVER 1 - UNROT 1. -  ^} { DROP2 } IFTE  » DOSUBS
 PILIST
»
```

92 Bytes << 98765430111 PHI >> TEVAL --> 61497371520 s: .34 (native EULER function is .12 s)

```
«
 FACTORS DUP
 1. « DUP TYPE NOT {DROP} IFT »  DOLIST
 SWAP
 1. « DUP TYPE {DROP} IFT »  DOLIST
 1 - OVER SWAP  ^ SWAP 1 - + PILIST
»
```

122 Bytes << 98765430111 PHI >> TEVAL --> 61497371520 s: .572

```
«
 FACTORS DUPDUP
 1. « TYPE » DOLIST
 DUP NOT ROT IFT
 1. -
 UNROT SWAP IFT
 DUP ROT ^ SWAP 1 - + PILIST
»
```

86,5 Bytes << 98765430111 PHI >> TEVAL --> 61497371520 s: .595

```
«
 FACTORS DUP SIZE 1.
  -> s p
  «
   EVAL
   1. s START
     OVER 1 -  UNROT 1. -  ^ *  'p' STO*
   2. STEP
```

p
»
»

100.5 Bytes << 98765430111 PHI >> TEVAL --> 61497371520 s: .29

I have to add the special case 1 test !

Edited: 2 June 2012, 9:07 a.m.

Re: Weekend programming challenge: Euler's Totient function

*Message #8 Posted by **Gerson W. Barbosa** on 2 June 2012, 12:54 p.m.,*
in response to message #7 by Gilles Carpentier

Hello Gilles,

> Quote:
>
> I dont understand why it works, but it works fine

Neither do I! I started with phi(p) = p - 1, then I examined the first few composites for a similar relationship. It wasn't difficult to find the pattern. Since the algorithm is based on an observation, rigorously it could not be used unless it had been proved first, a task I am definitely not up to.

Thanks for your great implementations of the algorithm! The latter takes 0.35 seconds on my HP 50g, 0.38 seconds when tested for 1, about twice the time it takes when using the built-in EULER function.

Gerson.

Edited per Gilles's observation below
Edited to remove a reference to Lehmer's conjecture, which does not apply in this case.

Edited: 2 June 2012, 2:57 p.m. after one or more responses were posted

Re: Weekend programming challenge: Euler's Totient function

*Message #9 Posted by **Gilles Carpentier** on 2 June 2012, 1:27 p.m.,*
in response to message #8 by Gerson W. Barbosa

I think you meant .35s and not 3.5s

I run a test at full speed emulation which compares with the native EULER function... It's now 100000 and all is perfect.I will see tomorrow!

interesting things :

http://mathworld.wolfram.com/TotientFunction.html

Edited: 2 June 2012, 1:54 p.m.

Re: Weekend programming challenge: Euler's Totient function

*Message #10 Posted by **Gerson W. Barbosa** on 2 June 2012, 2:14 p.m.,*
in response to message #9 by Gilles Carpentier

> Quote:
>
> I think you meant .35s and not 3.5s

Sorry for the mistake. I will fix it.

Gerson.

Re: Weekend programming challenge: Euler's Totient function

*Message #11 Posted by **Gerson W. Barbosa** on 2 June 2012, 6:29 p.m.,*
in response to message #9 by Gilles Carpentier

> Quote:
>
> I run a test at full speed emulation which compares with the native EULER function... It's now 100000 and all is perfect.

You might want to change your program slightly to allow for results greater than 999 999 999 999 and repeated evaluations:

```
%%HP: T(3)A(R)F(,);
\<< DUP 1 \=/
  IF
  THEN FACTORS 1 IQUOT DUP SIZE 1 \-> s p
    \<< EVAL 1 s
      START OVER 1 - UNROT 1 - ^ * 'p' STO* 2
      STEP p
    \>>
  END
\>>


<< 123456789123456789123456789 PHI >>       TEVAL  -->    821514062010415763062684800
                                                                          s:9.4735
<< 123456789123456789123456789 EULER >>     TEVAL  -->    821514062010415763062684800
                                                                          s:8.7739
```

Re: Weekend programming challenge: Euler's Totient function

*Message #12 Posted by **Valentin Albillo** on 2 June 2012, 1:07 p.m.,*
in response to message #4 by Ángel Martin

Hi, Ángel:

> Quote:
>
> Maybe I didn´t read the original posting well, but I think the challenge was about programming the PHI function - not about executing the function from machines that have it implemented.

Of course, that's a given, I read and interpreted the OP's request exactly that way.

But my point is that either the totient function is already easily available (be it directly built-in or in some module, library, or whatever), or else it just immediately and trivially reduces to and has the same complexity as finding the prime factors of its argument, so nothing really interesting to actually explore or improve upon, as finding prime factors on any and all handheld models and languages has already been beaten to death a zillion times by now.

> Quote:
>
> TOTNT is also available in my ¨ALGEBRA¨ module for the 41, and in the i41CX emulator. I used a brute-force approach that requires the prime factors of the argument, so nothing to write home about.

That's the point, finding prime factors yet again is nothing to write home about. On the other hand, this admittedly much worse but relatively novel approach does *not* require finding prime factors at all:

(one line of **HP-71B** code, RADIANS mode assumed):

```
10 DEF FNT(N) @ S=0 @ R=2*PI/N @ FOR K=1 TO N @ S=S+GCD(K,N)*COS(R*K) @ NEXT K @ FNT=IROUND(S) @ END DEF


  >FNT(36)
               12
  >FNT(45)
               24
  >FNT(46)
               22
  >FNT(1000)
               400
  >FNT(11111)
               10800
```

Best regards from V.

Extended challenge: Euler's Totient chains

*Message #13 Posted by **Oliver Unter Ecker** on 2 June 2012, 4:21 p.m.,*
in response to message #12 by Valentin Albillo

> Quote:
>
> ...so nothing really interesting to actually explore or improve upon...

Ok. So that those who have built-in Euler phi functions don't have to brave the weekend without a calculator challenge, here's an extended challenge that centers around the phi function.

It's an adapted version of beautiful problem #214 at Project Euler.

Challenge:
Find the sum of the lengths of all totient chains for all primes below 1000. The totient chain is obtained by repeatedly applying phi until 1 is obtained.

Example:

```
The primes below 10 are {2, 3, 5, 7}.
The chains are:
2: 2 -> 1
3: 3 -> 2 -> 1
5: 5 -> 4 -> 2 -> 1
7: 7 -> 6 -> 2 -> 1
The sum of the lengths of these chains is 1+2+3+3 = 9.
```

This can be implemented straight-forwardly, but there's also room for exploration, as there're beautiful optimizations that can be made.

Re: Extended challenge: Euler's Totient chains

*Message #14 Posted by **Valentin Albillo** on 2 June 2012, 5:04 p.m.,*
in response to message #13 by Oliver Unter Ecker

Hi, Oliver:

> Quote:
>
> Challenge:
> Find the sum of the lengths of all totient chains for all primes below 1000. The totient chain is obtained by repeatedly applying phi until 1 is obtained.

For reference purposes, this is the 3-line straightforward implementation in **Emu71** (free **HP-71B** emulator):

```
10 DESTROY ALL @ INPUT K @ SETTIME 0 @ S=0 @ P=2 @ REPEAT @ L=0 @ N=P
```

```
20 REPEAT @ L=L+1 @ N=PHI(N) @ UNTIL N=1 @ S=S+L
30 P=FPRIM(P+1) @ UNTIL P>K @ DISP "Sum of the lengths:";S,TIME;"seconds"


  >RUN
   ? 10
           Sum of the lengths: 9  in 0.05 seconds



  >RUN
   ? 1000

           Sum of the lengths: 1335  in 0.32 seconds
```

As for the original problem (*Project Euler #214*), I solved that one a couple of months ago. I won't give the answer here but it begins with 1 and ends in 3.

Thanks for your extended challenge and

Best regards from V.

Re: Extended challenge: Euler's Totient chains

*Message #15 Posted by **Oliver Unter Ecker** on 2 June 2012, 5:42 p.m.,*
in response to message #14 by Valentin Albillo

Wow. You're fast!

I confirm your result is correct. Nice implementation.

As you continue to be "under-challenged" and the weekend is still long, here's the special challenge just for you: speed it up by a factor of two!

(Your solution looks vaguely like my first solution in RPL+. I can speed up mine by a factor of 3 by using two (small) tricks. So, I assume the same should apply in your environment. No faster phi function required.)

If you don't want your computer to be beat by an iPhone running a sluggish interpreted language, the time to beat is 0.08s.

> Quote:
>
> As for the original problem (Project Euler #214), I solved that one a couple of months ago.

Cool. What language is your preferred one at PE? Are you aware of the RPL "team"? (Came into existence only last year.)

Cheers.

Re: Extended challenge: Euler's Totient chains

*Message #16 Posted by **Valentin Albillo** on 2 June 2012, 8:22 p.m.,*
in response to message #15 by Oliver Unter Ecker

Hi again, Oliver:

> Quote:
>
> Wow. You're fast!
>
> I confirm your result is correct. Nice implementation.

Thanks. Matter of fact you're way too kind, there's nothing nice about my posted implementation, it's just a plain-vanilla straightforward snippet of code just for reference purposes as it doesn't feature any clever tricks at all.

> Quote:
>
> As you continue to be "under-challenged" and the weekend is still long, here's the special challenge just for you: speed it up by a factor of two!

By a factor of two, you say ? Then this will do:

```
10 DESTROY ALL @ DIM F(1000) @ INPUT K @ SETTIME 0
20 S=1 @ P=3 @ REPEAT @ S=S+FNF(P-1) @ P=FPRIM(P+2) @ UNTIL P>K
30 DISP "Sum of the lengths:";S,TIME;""
40 !
50 DEF FNF(N) @ IF N=2 THEN FNF=2 @ END ELSE IF NOT F(N) THEN F(N)=FNF(PHI(N))+1
60 FNF=F(N)


   >RUN
    ? 1000


         Sum of the lengths: 1335  in 0.17 seconds
```

It's still a straightforward implementation devoid of any worthy mathematical shortcuts or clever programming but manages to run about twice as fast by using recursion and a cache.

> Quote:

> If you don't want your computer to be beat by an iPhone running a sluggish interpreted language, the time to beat is 0.08s.

As the hardware and software platforms are so utterly different there's no point to the comparison, it would be completely meaningless, so my above code's ~0.17 secs will have to do.

> Quote:
>
> What language is your preferred one at PE?

The one I used here (über-obsolete **HP-71B** interpreted **BASIC** as emulated on **Emu71** running on über-obsolete **MS-DOS**) is the one I use for every ***Project Euler*** problem. Anything else wouldn't be a challenge at all, thus no fun.

> Quote:
>
> Are you aware of the RPL "team"? (Came into existence only last year.)

I wasn't but that was to be expected as I absolutely and utterly *dislike* RPL (*'hate'* is tempting but probably too strong a word), which for me stands for **R**e**P**e**L**ent, **R**e**P**u**L**sive, **R**etardedly **P**reposterous **L**ingo, and assorted similar expressions. Thanks for the reference, though, you meant well and I really appreciate it.

Thanks again for your extended challenge and

Best regards from V.

Re: Extended challenge: Euler's Totient chains

*Message #17 Posted by **Oliver Unter Ecker** on 3 June 2012, 1:09 a.m.,*
in response to message #16 by Valentin Albillo

Hi Valentin,

Well, congrats on meeting the second challenge without resorting to math insight to achieve the speed-up. ;-)

> Quote:
>
> As the hardware and software platforms are so utterly different there's no point to the comparison...

I know. Was just teasing and smuggling in an extra-extra challenge for you...
Strangely enough, despite all the HW and SW diffs, both our original implementations had almost exactly the same elapsed time.

> Quote:
>
> ...absolutely and utterly dislike RPL...

Ok. Not to re-kindle the epic RPN vs RPL debate, but it seems to me that most simply prefer what they first got used to.

Cheers.

Re: Extended challenge: Euler's Totient chains

*Message #18 Posted by **Valentin Albillo** on 3 June 2012, 11:46 a.m.,*
in response to message #17 by Oliver Unter Ecker

> Quote:
>
> Well, congrats on meeting the second challenge without resorting to math insight to achieve the speed-up. ;-)

Thanks. I notice the ;-) you added. Math insight belongs where it's needed. Should you have requested the total length for primes below 10^12, say, math insight to get the times reasonable would've been in order.

For the present case, where a tiny code fragment is interpreted by a very slowly emulated old language running on obsolete SO and hardware and still gets the job done in 0.17 sec, that would be sheer overkill. Just rewriting it in C#, Java, Python, or any other modern language would run 1,000+ times faster, still without using math insight at all.

> Quote:
>
> Ok. Not to re-kindle the epic RPN vs RPL debate, but it seems to me that most simply prefer what they first got used to.

Not me. I only prefer the very best and have no attachment to inferior products out of nostalgia, just because they were the first I stumbled upon. The reasons for my utter dislike of RPL are far more rational and substantial but I've posted them a number of times in the past and this is neither the place nor the time for an encore.

Thanks again and have a nice Sunday evening.

Best regards from V.

Re: Extended challenge: Euler's Totient chains

*Message #19 Posted by **Oliver Unter Ecker** on 3 June 2012, 4:10 p.m.,*
in response to message #18 by Valentin Albillo

Quote:

Math insight belongs where it's needed.

Sure. The challenge should run appreciably slower on various real calcs and *there* the math insight may be needed. Plus, again, the two or so tricks are fun to find--and fun was meant to be the main point here.

Quote:

I only prefer the very best and have no attachment to inferior products out of nostalgia...

Can you elaborate? You previously smothered your setup with derogatives (ueber-obsolete...).

What do you consider the best (for these kinds of tasks), and how do you define "best"?

(May I also ask about the run-time of PE #214 in your Emu71 setup. This tasks appears to me to be one of the various PE tasks that are quite brutal. Finding all primes below 40e6 is time-consuming any way you slice it. Have you published your solution in the solution thread?)

A nice Sunday to you too.

Re: Extended challenge: Euler's Totient chains

*Message #20 Posted by **Valentin Albillo** on 3 June 2012, 6:21 p.m.,*
in response to message #19 by Oliver Unter Ecker

Hi again, Oliver:

Quote:

Can you elaborate? You previously smothered your setup with derogatives (ueber-obsolete...).

They aren't meant as "derogatives" but as hard facts: a slow 1984 BASIC interpreter running on MS-DOS is über-obsolete, period. Of course this doesn't mean that I don't actually like this particular setup nor does it mean that it's useless, far from it.

Quote:

What do you consider the best (for these kinds of tasks), and how do you define "best"?

These kinds of tasks ? Do you mean math/programming challenges Project Euler-style ? Well, Mathematica or Maple, of course. Python, Java, or C# fitted with the relevant multiprecision libraries and such would also do fine.

As for the definition of "best", it depends. If you're going for maximum speed and power then "best" is one thing. If you're going for maximum ease and clarity then "best" is another thing. And if you're going for a real challenge due to very serious handicaps and having fun struggling like mad just to try and deliver the goods, then "best" is the setup I'm presently using.

> Quote:
>
> (May I also ask about the run-time of PE #214 in your Emu71 setup.

Project Euler states that about one minute should be enough for most problems but I'm not deluding myself about what my setup can physically achieve so taking into account that it runs some 1,000 times slower than using even mediocre modern languages I'm cutting myself some slack and thus consider that anything around 1 hour is tantamount to having achieved the goal.

That's not to say that I don't strive for faster times if it's at all possible. Just for instance, PE 162 runs in just a few seconds. For PE #214 my timing was 19 min 52 sec.

> Quote:
>
> This tasks appears to me to be one of the various PE tasks that are quite brutal. Finding all primes below 40e6 is time-consuming any way you slice it.

That depends on how you go about it. First of all, the FPRIM function quickly delivers the next prime number and can be used for some sizable ranges. Second, the same way that you can use this or that function (say LN(X) or PHI(N)) without having to program it yourself by simply using built-in functionality or some module, library, DLL, or LEX binary, nothing precludes that you use a pre-calculated, pre-stored prime file on disk.

You compute it once, to 10^8 or 10^9, say, then get your primes from it as fast as you can read them, whenever you need them. Simple, logical, and efficient.

> Quote:
>
> Have you published your solution in the solution thread?)

No, I never visit the solution threads, not interested, this is a completely private endeavour. I just enter my computed result in PE, get the green Ok, and that's it, next problem please.

Thanks for your interest, it's been a pleasure.

Best regards from V.

Re: Extended challenge: Euler's Totient chains

*Message #21 Posted by **Oliver Unter Ecker** on 3 June 2012, 9:54 p.m.,*
in response to message #20 by Valentin Albillo

Hi Valentin,

Thanks for your answers.

> Quote:
>
> And if you're going for a real challenge due to very serious handicaps and having fun struggling like mad just to try and deliver the goods, then "best" is the setup I'm presently using.

That's both funny and logically sound, and settles the question.

> Quote:
>
> ...it runs some 1,000 times slower...

My quest with PE is a similar one, actually. I set myself the goal to solve half of the PE problems using an interpreted/code-morphing language running on an interpreted language running on a mobile device, which gets me at least a factor of 100 away from C/C++ on a desktop machine.

I'm using the chance to build a language (RPL+) and command set that will allow me to write what I consider elegant code on the user side.

> Quote:
>
> No, I never visit the solution threads, not interested, this is a completely private endeavour. I just enter my computed result in PE, get the green Ok, and that's it, next problem please.

Really? To me, a good part of the fun is seeing how others accomplished the tasks. I'm regularly amazed at approaches others are taking and learning from it. But then, my math knowledge is pitiful and a lot of learning is in order.

Cheers.

Re: Extended challenge: Euler's Totient chains

*Message #22 Posted by **Valentin Albillo** on 4 June 2012, 6:15 a.m.,*
in response to message #21 by Oliver Unter Ecker

Good morning, Oliver:

> Quote:
>
> > My quest with PE is a similar one, actually. I set myself the goal to solve half of the PE problems using an interpreted/code-morphing language running on an interpreted language running on a mobile device, which gets me at least a factor of 100 away from C/C++ on a desktop machine.
>
> In that case anything from 10 mins to one hour or two should be considered success. Using a handheld calculator (or emulator thereof) instead of C# running in a multicore desktop is already handicap enough to further compound the situation by requesting physically unrealizable times, so some fair scaling is definitely in order.
>
> Quote:
>
> > I'm using the chance to build a language (RPL+) and command set that will allow me to write what I consider elegant code on the user side.
>
> Good luck with that. But I'd (non-patronizingly) suggest that elegance should play second fiddle to efficiency, which for most PE problems is all but utterly mandatory.
>
> Quote:
>
> > *Quote: No, I never visit the solution threads, not interested, this is a completely private endeavour. I just enter my computed result in PE, get the green Ok, and that's it, next problem please.*
> >
> > Really? To me, a good part of the fun is seeing how others accomplished the tasks. I'm regularly amazed at approaches others are taking and learning from it.
>
> Really. I visited the solution threads just once right after solving my very first PE problem (PE 101) in order to see how the threads went, and after a cursory glance decided that I didn't want to see any of it lest I'd spoil both the fun *and* the learning.
>
> I'll explain: it's not unfrequent that I do manage to solve one of the problems yet I'm not fully satisfied with my approach. Later, I may revisit the problem and come out with a much better approach, all on my own. That additional satisfaction and the self-learning which comes with it would be completely ruined if I simply looked at the solution threads.
>
> Also, I'm on PE problems strictly for the fun, not to "learn" from others. Any and all learning will be self-taught, by working through a problem to and fro the hard way till I solve it to my complete satisfaction. That's how I learn the most, through sheer effort, and not only do I learn new techniques but also the fine art of finding worthy resources, new books, new articles. Simply looking at other people's solutions, which requires no effort whatsoever, absolutely pales in comparison and is but the easy, lazy approach which, as I said before, just spoils both the fun and the learning.
>
> Quote:

> But then, my math knowledge is pitiful and a lot of learning is in order.

I ver much doubt that all too modest assertion but indeed a certain level of math can do wonders for PE problems. Consider PE 276, for instance, which is a truly wonderful problem which can be enunciated in one line and everyone can understand exactly what is asked, yet a straight brute-force attack is doomed to failure from the start.

If people attempt such primitive approach they'll quickly find out that the problem is O(N^3), thus completely unmanageable. After some thinking and a little math reasoning it's possible to reduce it to O(N^2), which is millions of times faster, yet it would still take a number of months or years to arrive at a solution.

Then, if people's math foundations are solid and sound, they'll eventually find a way to reduce the complexity to O(N), which finally delivers a correct solution in reasonable times (mine was ~19 min). Even better times are still possible but that would be going for the A+.

So yes, improving your math skills is both a prerequisite for and a consequence of PE problem-solving.

Best regards from V.

Re: Extended challenge: Euler's Totient chains

*Message #23 Posted by **Oliver Unter Ecker** on 4 June 2012, 1:59 p.m.,*
in response to message #22 by Valentin Albillo

> Quote:
>
> Simply looking at other people's solutions, which requires no effort whatsoever, absolutely pales in comparison and is but the easy, lazy approach which, as I said before, just spoils both the fun and the learning.

And thus, nobly you speak, and I hang my head in shame that I ever looked at a solutions thread...

Thank you for explaining your stance. It all makes sense and is beautiful. But, truly, some of us will (sadly) never come up with the wonderful solutions, because we just don't have or "get" the math (you need some entry point to even begin your research). So peeking becomes the only way to witness them.

> Quote:
>
> ...suggest that elegance should play second fiddle to efficiency...

I'm a speed nut and spend far more time than I should on making things fast (while users of my app wait for me to improve functionality and graphing...), so I'm with you on the need for efficiency.

But, to me, elegance (and conciseness) of code is the top goal. I think it's actually *the* requirement for making writing code on a mobile device worthwhile, interesting, enjoyable.

I wish I could enjoy writing code on the 50g. But I don't. RPL feels too clunky (and the editor is horrible). (Yet, I'm basing my improved language attempt on RPL, looking at adding small ingredients that transform the language into something enjoyable.)

I'm a believer in the "calculator" form factor (quotes, because I wish to include large smartphones and small tablets) and a "device that assists thinking about math problems" that you can carry around. But software that realizes the potential this has, still has to be written. (And it must have an elegant, expressive user language.)

> Quote:
>
> Consider PE 276, for instance, which is a truly wonderful problem...

Thanks for the pointer, I'll check it out. And thank you for this exchange.

Cheers.

Re: Extended challenge: Euler's Totient chains

*Message #24 Posted by **Valentin Albillo** on 4 June 2012, 3:57 p.m.,*
in response to message #23 by Oliver Unter Ecker

You're welcome, same here. And best of lucks with your RPL+ goals.

Regards. V.

Re: Extended challenge: Euler's Totient chains

*Message #25 Posted by **Oliver Unter Ecker** on 3 June 2012, 2:56 a.m.,*
in response to message #13 by Oliver Unter Ecker

As Valentin posted his implementation and the result is known, here's my unoptimized implementation in RPL+:

```
<< 1000 primes
   { phi } { 1 == } doUntil
   NIP
   total
>>

primes: produces an array of primes up to the given number
phi: the Euler totient function; as Gilles told us, that's EULER on 50g
```

```
doUntil: special RPL+ function for component-wise conditional iteration of a code fragment over the elements of an array;
         takes a 2nd code fragment to decide when to stop iterating
NIP: doUntil produces two arrays: final values, iteration counts; this command discards former, as not needed here
total: sum of all array elements (here: iteration counts = chain lengths); equivalent to Sigma on 50g
```

Note: no loops; all array/list processing. List processing functions may be useful for a 50g implementation, too, but the varying iteration count may pose a problem.

Timing is not important for this challenge but, for completeness, this runs in 0.36s on an iPhone. (More importantly, this can be entered and developed painlessly on the device.)

I hinted at two math tricks that can speed this up considerably. These tricks can be inferred by staring at the right (non-obvious) spots on the wikipedia page for the Euler totient function, or--and that's the fun part of this challenge, I'm hoping--by (simple) exploration.

I don't want to discourage others to come up with other implementations and/or finding these tricks, so I'm going to post my optimized solution only on Sunday evening.

Re: Extended challenge: Euler's Totient chains

*Message #26 Posted by **Oliver Unter Ecker** on 3 June 2012, 11:30 p.m.,*
in response to message #25 by Oliver Unter Ecker

The expense of this code is in the phi function. Reducing the number of calls to this functions can speed things up.

Two math insights can be used to reduce the number of function invocations:

1) *phi(p) = p-1, if p is prime*

That is, since we're operating on primes as input, the first phi call can be traded for a decrement op. Not only does this remove 168 calls, but it removes the most expensive of the phi calls for each element. (The greater the input, the more expensive the phi call, just like factorization. And a prime is the worst case anyway.)

RPL+ making use of this insight:

```
<< 1000 primes
   decr
   { phi } { 1 == } doUntil
   NIP
   incr total decr
>>
Why "incr total decr": doing decr in line 2, we'll undercount the number of interations by one per element; hence, this incr here;
the first prime, 2, will have the terminal value going into the first iteration and will be overcounted; hence, the decr off the total
```

Specific to RPL+, there's a "recurse" keyword for doUntil that will terminate the **do** when a recursion is detected. Using this yields the following simplified code:

```
<< 1000 primes
   decr
   { phi } { recurse } doUntil
   NIP
```

```
   total
>>
```

2) Repeated application of phi will eventually produce a power of two value. Further, *phi(pOf2) = pOf2/2*. That is, the remaining length of the chain can be computed by simply taking the log2 of the first pOf2 value encountered:

```
<< 1000 primes
   decr
   { phi } { isPowerOf2 } doUntil
   SWAP log2 +
   incr total decr
>>
```

Like RPL, RPL+ has no types. But it has related optional hints for code fragments.
Changing

```
   { phi } { isPowerOf2 } doUntil
```

into

```
   { real phi } { real isPowerOf2 } doUntil
```

tells the compiler that Reals will be used for the function calls and permits it to optimize things.

With these things done, the code runs 4x faster.

The component-wise operation of **doUntil** is the main reason here why this code can be written much more compactly than the RPL code. It also has performance advantages. For example, the function internally caches intermediate results. (And it uses this cache to detect recursion virtually for free.)

Re: Extended challenge: What would be the "best" syntax for this?

*Message #27 Posted by **Oliver Unter Ecker** on 4 June 2012, 12:23 p.m.,*
in response to message #25 by Oliver Unter Ecker

(I promise this is my last post about this, if there're no responses. Hope to not be annoying.)

I'm curious: what do people think would be the "best" syntax for solving this problem? I'm asking strictly theoretical. You may make up your own language.

My suggestions (all of them non-existing fantasy languages; yet none of them completely satisfying):

```
sum = 0
for (p in primes < 1000)
   do p=phi[p]; sum++ until p==1
sum
(clear but not as concise as it could be)

iterate phi over primes < 1000; stop when 1
return sum of iteration counts
(clear but textual style lowers first glance comprehensibility)
```

```
primes<1000: do { phi } until { ==1 }; sum(iter_count)
(a bit cryptic)
```

I'd love to see a convincing example of something that looks, and is, clear but avoids variables. (For concise code.)

If someone knows Haskell (which I keep hearing about), I'd love to see an implementation.

Re: Extended challenge: What would be the "best" syntax for this?

*Message #28 Posted by **Gilles Carpentier** on 4 June 2012, 6:09 p.m.,*
in response to message #27 by Oliver Unter Ecker

Hi Oliver ! I'm very interested

```
sum = 0
for (p in primes < 1000)
   do p=phi[p]; sum++ until p==1
sum
(clear but not as concise as it could be)

«
  0. -> Total
  «
   2 « NEXTPRIME » « 1000 < » While
   1 « DO EULER 1. 'Total' STO+ UNTIL DUP 1 == END » DOSUBS
   Total
  »
»


or


«
  0.
  2 « NEXTPRIME » « 1000 < » While
  1 « DO  EULER SWAP 1. + SWAP UNTIL DUP 1 == END DROP » DOSUBS
»
```

Like every time the stack is used, you must try with an exemple to understand what the program do ...

About the 'best' syntax, i dont know ... I like this idea :

```
 «
 2 « NEXTPRIME » « 1000 < » While
 1 « « EULER » « 1 > » While SIZE » DOSUBS
 \SLIST
»
```

Which is nothing else that :

Give me the list of the primes below 1000
For each value how many iteration of EULER function to get 1 ?
Give me the sum of all these iterations

Something like (in my 'dream' (?) syntax):

```
{1..1000} «IsPrime?» Filter
« « Euler » « 1 > » CountWhile » ForEach
SList
```

About Haskell,there is a web site with many Euler problems examples below number 200

Edited: 4 June 2012, 6:46 p.m.

Re: Extended challenge: What would be the "best" syntax for this?

*Message #29 Posted by **Oliver Unter Ecker** on 4 June 2012, 7:35 p.m.,*
in response to message #28 by Gilles Carpentier

Thanks for these ideas, Gilles! I like your dream syntax.

I'll try out your RPL snippets later.

By the way, I implemented the "1..n" syntax in RPL+, based on your suggestion at comp.sys.hp48 from some time ago. Thank you for that! PE #5, for example, looks like this now: << 1..20 vlcm >>. You may also write << 1..1000 << ISPRIME? >> filter >>, as you suggest. You can also write [1..3 4..6 7..9] to build a matrix [[1 2 3][4 5 6][7 8 9]]. Variables are allowed instead of numbers: "1..n", etc.

Half of the things you suggest in the other thread are in RPL+ too, such as ability to write SIN(45) for 45 SIN. Syntactically, this was very easy: you just drop the quotes from an algebraic expression, and, voila, it becomes evaluated automatically. The end result is that you can mix both algebraic and post-fix styles in the same program.

I've looked at quite a bit of Haskell code but I keep scratching my head. I hear people raving about it but it also seems to be a language which requires digesting various concepts first before you can comprehend it. (Not necessarily a bad thing.)

Thanks for the Haskell pointer. I may look for that site.

Edited: 4 June 2012, 8:11 p.m.

Re: Extended challenge: What would be the "best" syntax for this?

*Message #30 Posted by **Gilles Carpentier** on 5 June 2012, 6:39 p.m.,*
in response to message #29 by Oliver Unter Ecker

Just to show the power of UsrRPL, a simple CountWhile implémentation :

```
« 0
   -> p t n
  «
   WHILE
     1 'n' STO+
     p EVAL DUP t EVAL
   REPEAT
   END
   DROP n
  »
»
'CountWhile' STO

 100 « EULER » « 1 > » CountWhile

-> 6

 2. « SQ  » « 100000. < » CountWhile

-> 5

 1. « 2. * 5. + 2. /  » « 1000. < » CountWhile

->400
```

Edited: 5 June 2012, 6:49 p.m.

HP50G solution with list processing

*Message #31 Posted by **Gilles Carpentier** on 4 June 2012, 5:11 p.m.,*
in response to message #25 by Oliver Unter Ecker

Hi Oliver

"Note: no loops; all array/list processing. List processing functions may be useful for a 50g implementation, too, but the varying iteration count may pose a problem."

I use a lot "Gofer list library" for this kind of things:

2 « NEXTPRIME » « 1000 < » While

Generate the list of the 1000 first prime number

http://www.musikwissenschaft.uni-mainz.de/~ag/hp49/README-GoferLists

o, a very compact program to solve the problem is :

```
«
 2 « NEXTPRIME » « 1000 < » While
 1 « « EULER » « 1 > » While SIZE » DOSUBS
 \SLIST
»


-> 1335  in 4 sec with emu48
```

Edited: 4 June 2012, 5:28 p.m.

Re: HP50G solution with list processing

*Message #32 Posted by **Oliver Unter Ecker** on 4 June 2012, 5:33 p.m.,*
in response to message #31 by Gilles Carpentier

Hi Gilles,

I see. Gofer list's "While" seems to do just about the same thing as RPL+'s "doUntil". Great. (Hey! That lib's from the university I went to...)

It's very nice to see that it's appreciably faster than non-list RPL, too.

Thanks! When I saw your title I thought you'd come up with a way to do it with the 50g's built-in list processing functions, which would have been a real surprise.

Cheers.

Re: Extended challenge: Euler's Totient chains

*Message #33 Posted by **Oliver Unter Ecker** on 3 June 2012, 11:02 p.m.,*
in response to message #13 by Oliver Unter Ecker

Here's an unoptimized RPL implementation for the 50g:

```
<<  0. -> sum <<
   1 @ init from primes
   WHILE
      NEXTPRIME
      DUP 1000. <
   REPEAT
      DUP
      0 -> length << @ compute chain length
         DO
            EULER DUP
```

```
        UNTIL
           1 ==
           'length' 1 STO+
        END
        DROP
        length 'sum' STO+
     >>
  END
  DROP
  sum
>> >>
```

This is as clean as I can make it, and with all the DUPs and DROPs it ain't pretty. If you see how to make it prettier and/or use list processing, please jump in.

Speed is not good either. 23s on Emu48 on a capable machine. (Approx. mode on or off; it doesn't matter.) It looks like NEXTPRIME and EULER are internally implemented as ZINT functions only, which carries a substantial speed penalty.

Re: Weekend programming challenge: Euler's Totient function

*Message #34 Posted by **Oliver Unter Ecker** on 2 June 2012, 4:22 p.m.,*
in response to message #3 by Gilles Carpentier

I also hadn't known the 50g had this function built-in. Thanks!

Re: Weekend programming challenge: Euler's Totient function

*Message #35 Posted by **Tim Wessman** on 3 June 2012, 10:12 a.m.,*
in response to message #34 by Oliver Unter Ecker

So does the 39gII.

Now granted, there isn't an exact ZINT object type or equivalent, so it is limited to floating point input.

TW

Re: Weekend programming challenge: Euler's Totient function

*Message #36 Posted by **Oliver Unter Ecker** on 2 June 2012, 4:42 p.m.,*
in response to message #1 by Allen

Here's an implementation in JavaScript:

```
"phi": function(n) {
        var result = 1;
        if (!(n%2)) { n /= 2; while (!(n%2)) { n /= 2; result *= 2; } }
        for (var k=3; k*k<=n; k+=2) {
                if (!(n%k)) {
                        n /= k;
```

```
                        result *= k-1;
                        while (!(n%k)) {
                                n /= k;
                                result *= k;
                        }
                }
        }
        if (n>1) result *= n-1;
        return result;
}
```

Change "var" to "int" and this should be valid C code.

I would like to see something faster.

As I searched for fast implementations, I found sieve-like methods that will speed up computing a range of phis. There's a method that pre-computes an array of factors that then can be used to more quickly compute individual values, for example.

The actual implementation in ND1 uses this to more quickly compute individual phis that fall in a "prepped" range.

Full code:

```
"prepPhi": function(M) {
        var fact = [];
        for (var i=0; i<M; i++) fact[i]=1;
        for (var i=2; i<M; i++) if (fact[i]==1) for (var k=i+i; k<M; k+=i) if (fact[k]==1) fact[k]=i;
        calculator.vars.phiF = fact;
},
"phi_prepped": function(x) {
        if (x <= 1) return 1;
        var f = calculator.vars.phiF[x];
        if (f == 1) return x-1;
        var p = 1, fp = 1;
        while ((x /= f) % f == 0) { p++; fp *= f; }
        return arguments.callee(x)*(f-1)*fp;
},
"phi": function(n) {
        if ("phiF" in calculator.vars && n < calculator.vars.phiF.length) return this.phi_prepped(n);
        ... /* as above */
},
```

Re: Weekend programming challenge: Euler's Totient function

*Message #37 Posted by **Paul Dale** on 3 June 2012, 10:39 p.m.,*
in response to message #1 by Allen

I did consider including this function in the 34S at one stage :-)