Extended EA in code

inverses

We want to find multiplicative inverses, and to do that we will use the extended Euclidean algorithm.

First, here is a brute-force method which manually checks the remainder for every `n` until it finds one.

If the inverse does not exist, this function will never return.

```
# requires a > b
def caveman(m,p):
    n = 2
    while m*n % p != 1:
        n += 1
    return n
```

The inverse exists if and only if the greatest common divisor is `1`. In the case of arithmetic `mod 60`, the integer `6` does *not* have a multiplicative inverse. That's because

```
6 * i mod 60
```

cannot be equal to `1`. For `mod 60` only the following numbers have inverses (as shown):

```
7 43
11 11
13 37
17 53
19 19
23 47
29 29
31 31
37 13
41 41
43 7
47 23
49 49
53 17
59 59
```

extended EA recap

In the previous write-up, we went through two methods for using the information from a run of the EA to compute the multiplicative inverse of a number mod `a`, the larger number input to the EA algorithm.

I call these the "forward" and "backward" methods.

The backward method runs the EA to completion first, then processes the data in reverse order. I found it relatively easy to code this one.

backward code

With data from a run for `a = 231` and `b = 130`:

```
101 = 231 - (1)130
 29 = 130 - (1)101
 14 = 101 - (3)29
  1 =  29 - (2)14
  0 =  14 - (1)14
```

Consider this step:

```
1 = 29 + (-2)14
1 = 29 + (-2)[101 - 3(29)]
1 = (-2)101 + (7)29
```

We need two variables, call them `x` and `y`.

The data from the previous round is `x = 1` (the coefficient of `29`), and `y = -2` (the coefficient of `14`).

For the update, we need the new value of `q`, the quotient in the equation `q(b) = (3)29`.

The new values of `x` and `y` are:

```
x = y
y = x - yq
```

where `x` in the second equation is the old value. We need a temporary variable to hold this value.

Here is the code for the inner loop from my version:

```
for i in range(2,N):
    a,b,q,r = L.pop(0)
    tmp = x
    x = y
    y = tmp - q * y
    t = a,b,q,r,x,y
    pp(t)   # pretty print
```

For the first step, the equation is:

```
1 =  29 - (2)14
```

so the values we need for `x` and `y` are:

```
x = 1
y = -q
```

Here is a run with some numbers I picked out of thin air:

```
> python eea.py 333337 58498
a=16,b=3,q=5,r=1,x=1,y=-5
a=35,b=16,q=2,r=3,x=-5,y=11
a=86,b=35,q=2,r=16,x=11,y=-27
a=465,b=86,q=5,r=35,x=-27,y=146
a=1016,b=465,q=2,r=86,x=146,y=-319
a=5545,b=1016,q=5,r=465,x=-319,y=1741
a=17651,b=5545,q=3,r=1016,x=1741,y=-5542
a=40847,b=17651,q=2,r=5545,x=-5542,y=12825
a=58498,b=40847,q=1,r=17651,x=12825,y=-18367
a=333337,b=58498,q=5,r=40847,x=-18367,y=104660
multiplicative inverse of 58498 is 104660 mod 333337
>
```

Check it:

```
>>> 58498 * 104660 % 333337
1
>>>
```

I found some code here

```
# return (g, x, y) a*x + b*y = gcd(x, y)
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)
```

What this recursive version does in effect is the backward algorithm. It recursively calls itself, then builds up the result through all the returns.

It's a bit confusing because the values of `b` and `a` are switched compared to my usage: it does `b % a`. `x` and `y` are also switched.

The lines

```
g, x, y = egcd(b % a, a)
return (g, y - (b // a) * x, x)
```

carry out the computation, assigning `x` for `y`, and `y - q*x` for `x`, in the `return` line.

forward code

Our run for `a = 231` and `b = 130` had this data:

```
101 = 231 - (1)130
 29 = 130 - (1)101
 14 = 101 - (3)29
  1 =  29 - (2)14
  0 =  14 - (1)14
```

and these steps:

```
101 = 231 - (1)130
    = a - b

29 = 130 - (1)101
   = b - (1)(a - b)
   = (-1)a + 2b

14 = 101 - (3)29
   = (a - b) - (3)[(-1)a + (2)b]
   = 4a - 7b

1 = 29 + (-2)14
  = (-1)a + 2b - (2)[4a - 7b]
  = -9a + 16b
```

After noodling around for a bit, here is the pseudocode I came up with. We need to retain values from two rounds back, stored in the variables `s` and `t`.

```
round -1
s = 0
t = 1

round 0
q = 1
x = 1
y = -q = -1

round 1
q = 1
tmp = x,y = 1,-1
x = s - qx = 0 - 1 = -1
y = t - qy = 1 - (1)-1 = 2
s, t = tmp = 1,-1

round 2
q = 3
tmp = x,y = -1,2
x = s - qx = 1 - 3(-1) = 4
y = t - qy = -1 - (3)2 = -7
s, t = tmp = -1,2

round 3
q = 2
tmp = x,y = 4,-7
x = s - qx = -1 - 2(4) = -9
y = t - qy = 2 - (2)(-7) = 16
s, t = tmp = 4,-7

y is the inverse
```

This is the inner loop:

```
while r != 0:
    a,b = b,r
    q = a / b
    r = a % b
    tmp = x,y
    x = s - q*x
    y = t - q*y
    s,t = tmp
    print 'x',x,'y',y,'s',s,'t',t
```

eea_forward.py

```
> python eea_forward.py 231 130
x 1 y -1
x -1 y 2 s 1 t -1
x 4 y -7 s -1 t 2
x -9 y 16 s 4 t -7
x 130 y -231 s -9 t 16
16 is the inverse of 130 mod 231
>
```

Check it

```
>>> 16 * 130 % 231
1
>>>
```

Here is code from the web to carry out the forward method. I haven't figured out how it works yet.

```
def eea(a,b):
    s, t = 1, 0
    u, v = 0, 1
    while b != 0:
        q = a / b
        a, b = b, a % b

        tmp = s, t
        s = u - (q * s)
        t = v - (q * t)
        (u,v) = tmp
    return u

print eea(53,10)
```

This is very similar to what I had, except that `u,v` plays the rols of `s,t`, and `s,t` plays the role of `x,y`.