Fast modular exponentiation, help me find the mistake

Asked 5 years, 9 months ago Modified 5 years, 9 months ago Viewed 4k times



I am trying to implement a scheme of fast exponentiation. Degree is represented in binary form:

4

2

```
def pow_h(base, degree, module):
    degree = bin(degree)[2:]
    r = 1

for i in range(len(degree) - 1, -1, -1):
        r = (r ** 2) % module
        r = (r * base ** int(degree[i])) % module
```

But function is not working properly, where is the mistake?

$$x^{\alpha} \pmod{n} = (\dots(\dots x^{\alpha_{k-1}})^2 x^{\alpha_{k-2}})^2 \dots x^{\alpha_1})^2 x^{\alpha_0} \pmod{n}$$

python algorithm

return r

Share Edit Follow Flag

edited Nov 13, 2016 at 20:38

OneCricketeer

1594 19 119 210

asked Nov 13, 2016 at 20:37



- add print() in function to see what values you get and compare with own calculations on paper.
- furas Nov 13, 2016 at 20:46
- FWIW, the built-in pow function accepts a modulus as an optional 3rd argument. PM 2Ring Nov 13, 2016 at 21:00

2 Answers





As I said in the comments, the built-in pow function already does fast modular exponentiation, but I guess it's a reasonable coding exercise to implement it yourself.

Your algorithm is close, but you're squaring the wrong thing. You need to square $_{\text{base}}$, not $_{\text{r}}$, and you should do it after the multiplying step.





```
def pow_h(base, degree, module):
    degree = bin(degree)[2:]
    r = 1
    for i in range(len(degree) - 1, -1, -1):
        r = (r * base ** int(degree[i])) % module
        base = (base ** 2) % module
    return r

#test

for i in range(16):
    print(i, 2**i, pow_h(2, i, 100))
```

output

```
0 1 1
1 2 2
2 4 4
3 8 8
4 16 16
5 32 32
6 64 64
7 128 28
8 256 56
9 512 12
10 1024 24
11 2048 48
12 4096 96
13 8192 92
14 16384 84
15 32768 68
```

Using r * base ** int(degree[i]) is a cute trick, but it's probably more efficient to use a if statement than exponentiation. And you can use arithmetic to get the bits of degree, rather than using string, although bin is rather efficient. Anyway, here's my version:

```
def pow_h(base, power, modulus):
    a = 1
    while power:
        power, d = power // 2, power % 2
        if d:
            a = a * base % modulus
        base = base * base % modulus
    return a
```

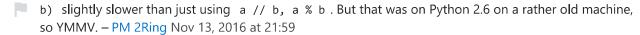
Share Edit Follow Flag

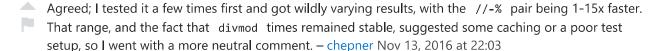
answered Nov 13, 2016 at 21:06



```
I might use power, d = divmod(power, 2) instead. – chepner Nov 13, 2016 at 21:26
```

@chepner: I occasionally use divmod for its elegance, but in my timeit tests I've found divmod(a,

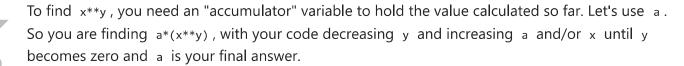


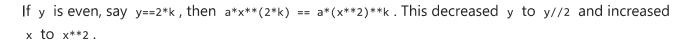




Such fast exponentiation must act differently if the current exponent is even or odd, but you have no such check in your code. Here are some hints:







If y is odd, say y==2k+1, then a*x**(2*k+1) == (a*x)*x**(2*k). This decreased y to y-1 and increased a to a*x.

You should be able to figure the algorithm from here. I did not include using the modulus: that should be easy.

Share Edit Follow Flag

answered Nov 13, 2016 at 20:49



Rory Daulton **20.9k** 5 39 48