Discrete logarithm

If g is a primitive root mod p (prime) nad $p \nmid h$ the sentence in Sage

```
log( Mod(h,p),  Mod(g,p) )
```

gives the discrete logarithm $0 \leq x < p-1$ such that $g^x \equiv h \pmod{p}$.

In order to simplify this heavy notation let us define a function:

```
# It works only when g is a primitive root modulo p
def dlog(h,g,p):
    return log( Mod(h,p),  Mod(g,p))
```

The output of

```
print primitive_root(103)
print dlog(8,5,103)
#of course dlog(0,5,103) does not exists
print dlog(1,5,103)
print dlog(5,5,103)
print dlog(22,5,103)
print dlog(13,5,103)
```

is `5, 30, 0, 1, 3, 72`.

If g is not a primitive root for $\mathbb{F}_p$ then the discrete logarithm may be not well defined.

```
print 'A primitive root=',primitive_root(23)
# 2 is not a primitive root modulo 23 then
# the following sentence raises an error
# print dlog(8,2,23)
# In fact this discrete logarithm is not
# uniquely defined
print Mod(2^3,23)
print Mod(2^14,23)
```

gives

```
A primitive root= 5
8
8
```

Note: In some versions of Sage sentences like `dlog(8,2,23)` do not raise an error and return a valid solution of $2^x \equiv 8 \pmod{23}$.

Discrete logarithm by brute force

```
# Discrete logarithm by brute force
# Example 21^x=29 (107)
for x in range(106):
    if Mod(21,107)^x == Mod(29,107):
        print 'x= ',x
        break
```

```
# Discrete logarithm for g^x=h (p) by brute force

def br_fr(h,g,p):
    for x in range(p-1):
        if Mod(g,p)^x == Mod(h,p):
            print 'x= ',x
            break
```

Comparing the performance with sage built-in function

```
time br_fr(7,3,2^19-1)
time dlog(7,3,2^19-1)
```

The number $2^{19}-1$ is prime and 7 is a primitive root.

```
x=  243983
Time: CPU 7.46 s, Wall: 7.50 s
243983
Time: CPU 0.00 s, Wall: 0.00 s
```

Baby-step and giant-step tables for $3^x \equiv 62\ (101)$

```
# Table baby-step giant-step for 3^x = 62 (101)
n = 10
g = Mod(3,101)
gn = Mod(3,101)^(-n)
print 'i', 'b', 'g'
for i in range(10):
    print i, g^i, 62*gn^i
```

i	0	1	2	3	4	5	6	7	8	9
3^i	1	3	9	27	81	41	22	66	97	89
$62(3^{-10})^i$	62	60	32	44	10	39	41	69	57	91

Baby-step giant-step algorithm

```
# Discrete logarithm for g^x=h (p) by baby-step giant-step

def baby_giant(h,g,p):
    baby = [1]
    giant = [h]
    n = 1+floor(sqrt(p-1))

    for i in range(1,n):
        baby.append( Mod(baby[i-1]*g,p) )

    g = Mod(g,p)^-n
    for j in range(1,n):
        giant.append( Mod(giant[j-1]*g,p) )

    for inters in set(baby).intersection( set(giant) ):
#        print 'i =', baby.index(inters)
#        print 'j =', giant.index(inters)
        print 'x =',  baby.index(inters)+n*giant.index(inters)
```

Checking performance

```
print '# decimal digits =', (2^19-1).ndigits()
print '# binary digits (bits) =', (2^19-1).ndigits(2)
time baby_giant(7,3,2^19-1)
time print dlog(7,3,2^19-1)
print ''
print '# decimal digits =', (2^31-1).ndigits()
print '# binary digits (bits) =', (2^31-1).ndigits(2)
time baby_giant(8,7,2^31-1)
time print dlog(8,7,2^31-1)
```

```
# decimal digits = 6
# binary digits (bits) = 19
x = 243983
Time: CPU 0.08 s, Wall: 0.10 s
243983
Time: CPU 0.00 s, Wall: 0.00 s

# decimal digits = 10
# binary digits (bits) = 31
x = 1454746986
Time: CPU 4.41 s, Wall: 4.51 s
1454746986
Time: CPU 0.00 s, Wall: 0.00 s
```