Baby step
Giant step
Algorithm

THE TEAM

Editing & PPT

Nithin S

Theory

Jivan Prasad
Rutwik Krishna

Examples

Nived
Vihal Roy

Derivation

Sudihr
Nishant

Python Code

Ram Surya
Manbendra

Uses In Cryptography

Sudeep

DISCRETE LOGARITHM PROBLEM

The Discrete logarithms problem (DLP) can be described as follows:

$$\left.\begin{array}{ll} \text{Input}: & a, b, n \in \mathbb{N} \\ \text{Output}: & x \in \mathbb{N} \text{ with } a^x \equiv b \pmod{n} \\ & \text{if such an } x \text{ exists,} \end{array}\right\}$$

where the modulus n can either be a composite or prime.

For example, if the group is Z5*, and the generator is 2, then the discrete logarithm of 1 is 4 because $2^4 \equiv 1 \bmod 5$.

One of the most famous and generic algorithms
dealing with the discrete logarithm problem is
the so-called Baby-Step Giant-Step algorithm which is introduced
by Shanks

Let G be a finite cyclic group of order n, a 'a' generator of G and b belongs to G. The obvious algorithm for computing successive powers of a until b is found takes O(n) group operations. For example, to compute,

$$x = \log_2 15 \pmod{19}$$

we compute 2^x mod 19 for x = 0,1,2....19-1 until 2^x mod 19 = 15 for some x is found, that is

x	0	1	2	3	4	5	6	7	8	9	10	11
a^x	1	2	4	8	16	13	7	14	9	18	17	15

So, $\log_2 15 \pmod{19} = 11.$

ALGORITHM

This algorithm computes the discrete algorithm x of y to the base a, modulo n, such that y = a^x (mod n).

1] Initialization Computes $s = \lfloor \sqrt{n} \rfloor$.

2] Computing the baby step Compute the first sequence (list), denoted by S, of pairs (ya^r, r), $r = 0, 1, 2, 3, \cdots, s-1$:

$$S = \{(y, 0), (ya, 1), (ya^2, 2), (ya^3, 3), \cdots, (ya^{s-1}, s-1) \bmod n\}$$

and sort S by ya^r, the first element of the pairs in S.

3] Computing the giant step Compute the second sequence (list), denoted by T, of pairs (a^{ts}, ts), $t = 1, 2, 3, \cdots, s$:

$$T = \{(a^s, 1), (a^{2s}, 2), (a^{3s}, 3), \cdots, (a^{s^2}, s) \bmod n\}$$

and sort T by a^{ts}, the first element of the pairs in T.

4] Searching, comparing and computing Search both lists S and T for a match $ya^r = a^{ts}$ with ya^r in S and a^{ts} in T, then compute $x = ts - r$. This x is the required value of $\log_a y \pmod n$.

Let us take a small example :

Example: Suppose we wish to compute the discrete logarithm $x = \log_2 6 \bmod 19$ such that $6 = 2^x \bmod 19$.

STEP -1

$[1]\ y = 6,\ a = 2$ and $n = 19,\ s = \lfloor \sqrt{19} \rfloor = 4.$

STEP -2

[2] Computing the baby step:

$$
\begin{aligned}
S &= \{(y,0),(ya,1),(ya^2,2),(ya^3,3) \mod 19\} \\
&= \{(6,0),(6\cdot 2,1),(6\cdot 2^2,2),(6\cdot 2^3,3) \mod 19\} \\
&= \{(6,0),(12,1),(5,2),(10,3)\} \\
&= \{(5,2),(6,0),(10,3),(12,1)\}.
\end{aligned}
$$

STEP -3

[3] Computing the giant step:

$$
\begin{aligned}
T &= \{(a^s, s), (a^{2s}, 2s), (a^{3s}, 3s), (a^{4s}, 4s) \bmod 19\} \\
&= \{(2^4, 4), (2^8, 8), (2^{12}, 12), (2^{16}, 16) \bmod 19\} \\
&= \{(16, 4), (9, 8), (11, 12), (5, 16)\} \\
&= \{(5, 16), (9, 8), (11, 12), (16, 4)\}.
\end{aligned}
$$

STEP -4

[4] Matching and computing: The number 5 is the common value of the first element in pairs of both lists S and T with $r = 2$ and $st = 16$, so $x = st - r = 16 - 2 = 14$. That is, $\log_2 6 \pmod{19} = 14$, or equivalently, $2^{14} \pmod{19} = 6$.

USES

- When the modulus is a prime number, the Baby-step giant-step technique is frequently used to find the shared key in the Diffie Hellman key exchange.
- The Pohlig-Hellman algorithm solves the same problem with a lower algorithmic complexity if the modulus is not prime.
- By using baby step giant step algorithm, we can reduce the problem's complexity by storing the square root of the prime number used.
- Unfortunately, as our prime numbers grow larger, so does the amount of memory we require. As a result, discrete logarithms remain a challenging task for a computer.

PYTHON CODE

```python
from math import ceil, sqrt


def bsgs(g, h, p):

    '''
    Solve for x in h = g^x mod p given a prime p.
    If p is not prime, you shouldn't use BSGS anyway.
    '''
    N = ceil(sqrt(p - 1))  # phi(p) is p-1 if p is prime

    # Store hashmap of g^{1...m} (mod p). Baby step.
    tbl = {pow(g, i, p): i
    for i in range(N)}

    # Precompute via Fermat's Little Theorem
    c = pow(g, N * (p - 2), p)

    # Search for an equivalence in the table. Giant step.
    for j in range(N):
        y = (h * pow(c, j, p)) % p
        if y in tbl:
            return j * N + tbl[y]

    # Solution not found
    return None


print( " The Baby-Step-Giant-Step is used to solve for x in h = g^x mod p where p is a prime number. " )
print( " If p is not prime , then the Baby-Step-Giant-Step cannot be used. " )


g = int(input( " Enter the value of g : " ))
h = int(input( " Enter the value of h : " ))
print( " p should be a prime number. " )
p = int(input( " Enter the value of p : " ))

if p>1:
    for i in range(2,p):
        if (p%i)==0:
            print( " The entered value of p = " + str(p) + " is not a prime number.")
            break
        else:
            print( " The entered value of p = " + str(p) + " is a prime number.")
            x = bsgs(g,h,p)
            print( " Hence , the value of x is " + str(x) + "." )
```

Thank you!