Shanks' baby-step giant-step

June 4, 2022

1 Shanks' baby-step giant-step

- Alef Pinto Keuffer, a91683
- Beatriz Fernandes Oliveira, a91640
- Catarina Martins Sá Quintas, a91650

1.1 Baby-step giant-step algorithm for computing discrete logarithms

As described (slightly paraphrased) at [1]:

> Let $m = \lceil\sqrt{n}\rceil$, where n is the order of α. The baby-step giant-step algorithm is a time-memory trade-off of the method of exhaustive search and is based on the following observation. If $\beta = \alpha^x$, then one can write $x = im + j$, where $0 \le i, j < m$. Hence, $\alpha^x = \alpha^{im}\alpha^j$, which implies $\beta(\alpha^{-m})^i = \alpha^j$. This suggests the following algorithm for computing x.
>
> ---
>
> **INPUT**: a generator α of a cyclic group G of order n, and an element $\beta \in G$.
>
> **OUTPUT**: the discrete logarithm $x = \log_\alpha \beta$.
>
> 1. Set $m \leftarrow \lceil\sqrt{n}\rceil$.
> 2. Construct a table with entries (j, α^j) for $0 \le j < m$. Sort this table by second component. (Alternatively, use conventional hashing on the second component to store the entries in a hash table; placing an entry, and searching for an entry in the table takes constant time.)
> 3. Compute α^{-m} and set $\gamma \leftarrow \beta$.
> 4. For i from 0 to $m-1$ do the following:
>   Check if γ is the second component of some entry in the table.
>   If $\gamma = \alpha^j$ then return $(x = im + j)$.
>   Set $\gamma \leftarrow \gamma \cdot \alpha^{-m}$.
>
> ---

The above description gives us the following python implementation:

[25]:
```
def shanks_baby_step_giant_step(a,b,n):
    """
    Implements baby-step giant-step algorithm for computing discrete logarithms␣
 ↪as described at
```

```
    A. J. Menezes, S. A. Vanstone, and P. C. Van Oorschot, Handbook of applied␣
↪cryptography, 5th ed. CRC Press, 2001. p. 105.

    Parameters:
    a,b   G where G is a multiplicative group
    n is the order of G

    Returns: log_a(b)

    Complexity:
    Under the assumption that a group multiplication takes no more time than lg␣
↪n comparisons,
    the running time is O(√n).

    Requires storage for O(√n) group elements.
    """
    m = ceil(sqrt(n))
    table = {a^j:j for j in range(0,m)}
    y = b
    for i in range(0,m):
        if y in table:
            return i*m + table[y]
        y *= a^(-m)
```

1.1.1 Examples

With multiplicative groups $\mathbb{Z}_p^*$

[31]:
```
from sage.misc.html import MathJax
mj = MathJax()
from IPython.display import display, Math
from random import randrange
```

[30]:
```
n = 113
Zn = IntegerModRing(n)
 = Zn(3)
 = Zn(57)
log_  = shanks_baby_step_giant_step( , ,Zn.unit_group().order())
display(Math(rf"\left({  = },{  = },G = Z_{{n}}^*\right):\quad \log_{{ }}{ }␣
↪= {log_ }"))
```

[30]: $(\alpha = 3, \beta = 57, G = Z_{113}^*): \quad \log_3 57 = 100$

[62]:
```
def generate_example(max_prime=10000):
    n = random_prime(max_prime)
    Zn = IntegerModRing(n)
      = Zn.multiplicative_generator()
      = Zn(randrange(1,n))
```

```
    log_  = shanks_baby_step_giant_step( , ,Zn.unit_group().order())
    display(Math(rf"\left({  = },{  = },G = Z_{{{n}}}^*\right):\quad␣
↪\log_{{{ }}}{ } = {log_ }   { }^{{{log_ }}} = { }"))
```

[63]:
```
generate_example()
```

[63]: $(\alpha = 2, \beta = 2442, G = Z_{9661}^*) : \quad \log_2 2442 = 2707 \iff 2^{2707} = 2442$

[49]:
```
generate_example()
```

[49]: $(\alpha = 10, \beta = 1215, G = Z_{3313}^*) : \quad \log_{10} 1215 = 1883 \iff 10^{1883} = 1215$

2 References

[1] A. J. Menezes, S. A. Vanstone, and P. C. Van Oorschot, Handbook of applied cryptography, 5th ed. CRC Press, 2001. p. 105.