CHAPTER 4

Baby-Step Giant-Step Methods

1. Element order computation

We explain the general idea of Shanks' baby-step giant-step method [**Sha71**] with its first application, which is computing the order of an element. Let G be a finite group, and let $g \in G$. Assume that we know an upper bound E for $\operatorname{ord} g$. Let $q = \lceil \sqrt{E} \rceil$. Then we can write

$$\operatorname{ord} g = bq + s \ ,$$

with

$$0 \le b \le q \quad \text{and} \quad 0 \le s < q \ .$$

The numbers b and s are determined as follows. First, for $r = 0, 1, \ldots, q-1$ one computes the group elements g^{-r}. When doing this, for each r one checks whether $g^{-r} = 1$. If this is the case, we can set $\operatorname{ord} g = r$ and stop. Otherwise, these elements are stored in a table R; hence,

$$R = \{1, g^{-1}, g^{-2}, \ldots, g^{-(q-1)}\} \ .$$

These are the *baby steps*. Then for $a = 1, 2, \ldots, q$ one computes g^{aq}. These are the *giant steps*. For each a, one checks whether $g^{aq} \in R$. As soon as this happens, $\operatorname{ord} g$ is found, since then

$$g^{aq+r} = 1$$

for some $r \in \{0, 1, \ldots, q-1\}$, and $aq + r$ is the smallest positive number for which this is possible. Thus, set $b = a$ and $s = r$.

Here is the algorithm:

ALGORITHM 4.1.

This algorithm computes the order of the group element g in G.

INPUT: $g \in G$, upper bound E for $\operatorname{ord} g$
OUTPUT: $x = \operatorname{ord} g$

```
(1)  x = 0
(2)  if (g == 1 ) then
(3)    return (x)
(4)  fi
(5)  a = 1_G; f = g^{-1}
(6)  q = ⌈√E⌉
(7)  R = {(a, 0)}
(8)  for (r = 1, 2 ..., q − 1) do                    /* baby steps */
(9)    a = a * f
(10)   if (a == 1 ) then
(11)     x = r
(12)     return (x)
(13)   else
(14)     R = R ∪ {(a, r)}
(15)   fi
(16) od
(17) y = q; b = g^y; c = b
(18) while (x == 0) do                               /* giant steps */
(19)   if (there is a number r such that (b, r) ∈ R ) then
(20)     x = y + r
(21)     return (x)
(22)   else
(23)     y = y + q
(24)     b = b * c
(25)   fi
(26) od
```

We immediately see the following: If $\operatorname{ord} g \geq q$ $(= \lceil \sqrt{E} \rceil)$, Algorithm 4.1 requires one inversion and $q + \lfloor \operatorname{ord} g/q \rfloor + O(\log q)$ group multiplications. One has q group elements to store. It requires $\lfloor \operatorname{ord} g/q \rfloor$ table look-ups in a table of size q.

If the group elements are encoded as unique binary strings (what we assume in our generic setting), hashing on the group elements is possible so that the table-look ups can be implemented efficiently.

2. Discrete logarithm computation

To solve the discrete logarithm problem (given two elements g and h, where g serves as base), we modify Algorithm 4.1 as follows. As before, we assume that $E \geq \operatorname{ord} g$, and we put $q = \lceil \sqrt{E} \rceil$. We assume first that we do *not* know $\operatorname{ord} g$.

We compute the order of g in G as in Algorithm 4.1, i.e., we try to find integers y and r such that $g^{y+r} = 1$. However, for each y, before checking whether $g^{y+r} = 1$, we check whether $g^{y+r} = h$. For this, we work with the same set R as in Algorithm 4.1. We first check whether $(h^{-1} * g^y, r) \in R$, with R as in Algorithm 4.1. If this is the case, $\log_g h = y + r$. Otherwise, we check whether $(g^y, r) \in R$. As soon as we have found $\operatorname{ord} g$, we know that there is no discrete logarithm of x for base g, since $\log_g x < \operatorname{ord} g$. Just as in Algorithm 4.1,

during the computation of $R_0 = \{(g^{-r}, r) \ : \ 0 \le r \le q-1\}$ we always check whether $\log_g h$ or $\operatorname{ord} g$ is already found before we include a pair (g^{-r}, r).

ALGORITHM 4.2.

This algorithm computes the discrete logarithm of h $(h \neq 1)$ for base g.

INPUT: $h, g \in G$ $(h \neq 1)$, upper bound E for $\operatorname{ord} g$
OUTPUT: $t = 1$ and $x = \log_g h$ if $h \in \langle g \rangle$,
$t = 0$ and $x = \operatorname{ord} g$ if $h \notin \langle g \rangle$

```
(1)  x = 0
(2)  q = ⌈√E⌉; y = q
(3)  f = g^{-1}
(4)  a = 1; b = g^y; c = b                          /* a = g^{-r}, b = g^y,
(5)  R = {(a, 0)}                                   c = g^q  */
(6)  i = h^{-1}
(7)  for (r = 1, 2 ..., q − 1) do                   /* baby steps  */
(8)    a = a * f
(9)    if (a == i ) then
(10)     x = r; t = 1
(11)     break for
(12)   else
(13)     if (a == 1) then
(14)       x = r; t = 0
(15)       break for
(16)     fi
(17)   fi
(18)   R = R ∪ {(a, r)}
(19) od
(20) while (x == 0) do                              /* giant steps  */
(21)   if (there is a number r such that (i * b, r) ∈ R) then   /* checking for
(22)     x = y + r; t = 1                           discrete log.  */
(23)   else
(24)     if (there is a number r such that (b, r) ∈ R) then     /* checking for
(25)       x = y + r; t = 0                         the order of g  */
(26)     else
(27)       y = y + q
(28)       b = b * c
(29)     fi
(30)   fi
(31) od
(32) return (t, x)
```

Algorithm 4.2 requires two inversions and $q + 2\lfloor x/q \rfloor + O(\log q)$ group multiplications, and one has q elements to store. It requires $2\lfloor x/q \rfloor$ table look-ups in a table of size q.

If the order of g is known, we can work with the set $R_h = \{(h * a, r) \ : a \in R\}$ instead, which we initialize as $R_h = \{(h, 0)\}$. The next group elements to be stored in R_h are obtained then just by repeated multiplication with $f = g^{-1}$. This has the advantage that in the giant-step phase we only have to check for $(b, r) \in R_h$ rather than $(h * b, r) \in R$, which saves up to q multiplications. In order to detect when $h \notin \langle g \rangle$, we have to

check whether $y+q \le \operatorname{ord} g$ before doing the next giant step; if $y+q > \operatorname{ord} g$, we know that $h \notin \langle g \rangle$ and stop.

3. When no good upper bound is known

It is possible to remove the assumption that an upper bound of the element order is known and to obtain complexity results relative to the actual group order and the discrete logarithm rather than relative to an upper bound.

3.1. The BJT-method: dynamic doubling of the stepwidth([BJT97]). This method is based on the following statement:

LEMMA 4.1. *Let v be an even positive integer. For every positive integer x there are uniquely determined integers k, q and r with $k \ge 0$, $\lfloor 4^{k-1} \rfloor v^2 \le 2^k vq < 4^k v^2$ and $1 \le r \le 2^k v$ such that $x = y + r$, $y = 2^k vq$.*

PROOF. Let $x \in \mathbb{N}$. We first show the existence of such k, q and r. We choose k such that $\lfloor 4^{k-1} \rfloor v^2 < x \le 4^k v^2$, and write $x = 2^k vq + r$ with $1 \le r \le 2^k v$. Then $\lfloor 4^{k-1} \rfloor v^2 - 2^k v < 2^k vq \le 4^k v^2 - 1$, which implies that $2^k vq < 4^k v^2$. Moreover, if $k = 0$, we have $-v < vq$, so $0 \le vq$. If $k \ge 1$, we have $4^{k-1} v^2 - 2^k v < 2^k vq$; hence $2^{k-1}(v/2) - 1 < q$. Since $v/2$ is integral, this implies that $2^{k-1}(v/2) \le q$, so that $4^{k-1} v^2 \le 2^k vq$.

To show the uniqueness of this representation, let $x = 2^k vq + r$ with k, q and r as stated above. Then $\lfloor 4^{k-1} \rfloor v^2 \le 2^k vq < 4^k v^2$, which implies $q \le 2^k v - 1$, so that $x = 2^k vq + r \le 2^k v(2^k v - 1) + 2^k v = 4^k v^2$. Moreover, we have $\lfloor 4^{k-1} \rfloor v^2 < 2^k vq + r = x$. These inequalities determine k uniquely. The uniqueness of q and r is due to the uniqueness statement for division with remainder. □

We explain the method for computing $x = \operatorname{ord} g$. We select an even positive integer v which is used as the initial step-width in the algorithm. Then there is a unique non-negative integer k such that x belongs to the interval

$$I_k = \{\lfloor 4^{k-1} \rfloor v^2 + 1, \ldots, 4^k v^2\}.$$

We search those intervals for $k = 0, 1, 2, \ldots$ until x is found. By Lemma 4.1, each number in I_k can be written as $y + r$ with $y = 2^k vq$ and r, q as stated in Lemma 4.1. Also, each integer that can be written in this way belongs to the interval I_k. To check whether x is in I_k we test whether $g^{y+r} = 1$ with $y = 2^k vq$ and r and q as stated in Lemma 4.1. This means that we test whether

$$g^y = g^{-r}, \qquad 1 \le r \le 2^k v, \quad y = 2^k vq, \quad \lfloor 4^{k-1} \rfloor v^2 \le y < 4^k v^2.$$

For this purpose, we compute the set

$$R_k = \{(g^{-r}, r) : 1 \le r \le 2^k v\},$$

and for all values of q such that $\lfloor 4^{k-1} \rfloor v^2 \le y < 4^k v^2$ we check whether there exists $(g^y, r) \in R_k$ for some r. If so, $\operatorname{ord} g = y + r$. Otherwise, we increase k by 1. If $x \le v$, the set R_0 contains at least one pair $(1, r)$, and $\operatorname{ord} g = r$ for the smallest such r. Therefore, before adding a pair (g^{-r}, r), $1 \le r \le v$, to R_0 in the course of the computation of R_0, we always check whether $g^{-r} = 1$, and we break if the answer is "yes", since then $\operatorname{ord} g$ is already found.

The efficiency of the algorithm can be improved if we know a lower bound B of $\operatorname{ord} g$. Writing $C = B - 1$, we then work with the set $R_k = \{(g^{-(r+C)}, r) : 1 \le r \le 2^k v\}$, and if we find $(g^y, r) \in R_k$, $\operatorname{ord} g = y + r + C$. If no lower bound for $\operatorname{ord} g$ is known, we set $C = 0$.

We now present the algorithm.

ALGORITHM 4.3.

This algorithm computes the order of the group element g in G.

INPUT: $g \in G$, lower bound $C+1$ for $\operatorname{ord} g$, initial step-width v ($v \in 2\mathbb{N}$)
OUTPUT: $x = \operatorname{ord} g$

```
(1)  x = 0
(2)  s = 1; y = v; u = v
(3)  h = g^{-1}
(4)  a = h^C; b = g^v; c = b                    /* a = g^{-C-r};
(5)  R = ∅                                      b = g^y; c = g^u */
(6)  while (x == 0) do
(7)    for (r = s, s+1 ..., u) do               /* new baby steps */
(8)      a = a * h
(9)      if ( s == 1 ) then                     /* check if 1 ≤ x ≤ v */
(10)       if (a == 1 ) then
(11)         x = r + C
(12)         return (x)
(13)         break while
(14)       else
(15)         R = R ∪ {(a, r)}
(16)       fi
(17)     else
(18)       R = R ∪ {(a, r)}
(19)     fi
(20)   od
(21)   while (x == 0 and y < u^2) do             /* giant steps */
(22)     if (there is a number r such that (b, r) ∈ R ) then
(23)       x = y + r + C
(24)       return (x)
(25)     else
(26)       y = y + u
(27)       b = b * c
(28)     fi
(29)   od
(30)   s = u + 1; u = 2u                        /* double
(31)   c = c^2                                  step width */
(32) od
```

THEOREM 4.1. *Let $C = 0$. Let $x = \operatorname{ord} g$. For every choice of v, Algorithm 4.3 executes one inversion and at most $2\lfloor \log v \rfloor + 1$ multiplications in G and requires space for three group elements. On further group multiplications, space required, and table look-ups, we have the following estimates.*

1. *If $x \le v$, Algorithm 4.3 executes x additional multiplications in G. It uses a table of $x-1$ pairs $(g, r) \in G \times \{1, \ldots, x-1\}$ and it performs x equality checks.*
2. *If $\sqrt{x} \le v < x$, the number M of additional multiplications in G satisfies*

$$v \le M \le \lceil \sqrt{x} \rceil + v - 2.$$

The algorithm uses a table of v pairs $(g, r) \in G \times \{1, \ldots, v\}$, and it performs v equality checks. The total number TL of table look-ups satisfies

$$1 \le TL \le \lceil \sqrt{x} \rceil - 1.$$

3. *If $\sqrt{x} > v$, the number M of additional multiplications in G satisfies*

$$\frac{5}{4}\lceil\sqrt{x}\,\rceil + \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil - 1 \leq M \leq 4\lceil\sqrt{x}\,\rceil - \frac{v}{2} + \left\lceil \log \frac{\sqrt{x}}{v} \right\rceil - 5.$$

It performs v equality checks. It uses a table of at least $\lceil\sqrt{x}\,\rceil$ and at most $2\lceil\sqrt{x}\,\rceil - 2$ pairs $(g, r) \in G \times \{1, \ldots, 2\lceil\sqrt{x}\,\rceil\}$. The total number TL of table look-ups satisfies

$$\frac{\lceil\sqrt{x}\,\rceil}{4} \leq TL \leq 2\lceil\sqrt{x}\,\rceil - \frac{v}{2} - 2.$$

PROOF. See [**BJT97**]. □

REMARK 4.1. To adapt Theorem 4.1 to the case $C \geq 1$, we just have to replace each x by $x - C$ and add to the total number of group multiplications the multiplications required to compute $a = (g^{-1})^C$, i.e., at most $2\lfloor \log C \rfloor + 1$ multiplications.

As we see from Theorem 4.1, the efficiency of Algorithm 4.3 depends largely on the appropriate choice of the initial step-width v. As noted by Shanks [**Sha71**], the optimal choice of v is $v = \sqrt{\operatorname{ord} g}$. This results in about $2\sqrt{\operatorname{ord} g}$ group multiplications in our algorithm. If v is chosen too large (in comparison with $\sqrt{\operatorname{ord} g}$), we waste space and time because the set R is too big. If v is chosen too small, we waste time because of superfluous iterations of the outer while loop.