Memoryless generic discrete logarithm computation in an interval using kangaroos

Anthony MARTINEZ, Nicolas VALLET

January 2022

Table des matières

Synthèse

1 Compilation and execution instructions

1.1 Compilation

Since we are using some math functions, to compile our code, one must use the following command :
gcc main.c -lm

1.2 Execution

To execute the program, one must use the following command :
./a.out

2 Question 1

If we try to find the discrete logarithm by computing all possible values in the group G for example with the following algorithm :

```
Given h ang g such that h = g^a over G we want to find a
currentVal = g
for i in range 1, N with N the order of G:
 currentVal *= g
 if currentVal == h:
    return i
```

For thoose algorithm we have at most N multiplication over G. So we are in $O(N) = O(2^s)$ with s the size of the input so the complexity will be exponential which means that this would not be feasible on a personal computer.

3 Question 2

For this question we have used the fast exponentiation algorithm to implement the map $x \mapsto g^x$ with g = 4398046511104. The file testQuestion2.c test the function on the values given in the subject. Here is the algorithm :

```
exp(n, a): #with n the number and a the exponent
    result = 1
    while(a):
        if a&1
            result = result * n
        a >> = 1
        n = n * n
    return result
```

4 Question 3

For this question here is what we found :
Let's call A and B the number we want to multiply modulo $(2^{115} - 85)$.
Now let's write A and B as a concatenation of number for example 320 is the concatenation of 3 and 20. So let's take 4 numbers x, y, v, w and we write A = xy and B = wv where xy is the concatenation of x and y (same for wv).
Then we can write the multiplication as follow xy * wv = (x_ + y) * (w_ + v) = (x_ * w_) + (x_ * v) + (y * w_) + (y*v) where the symbol _ is a padding of 0. For example 320 * 110 = (300 + 20) * (100 + 10).
Therefore, to do our multiplication we have 4 additions to perform modulo $(2^{115} - 85)$.
Now we switch in the binary representation and we say that xy, wv are of length 128 bits at most and each number x, y, w, v are at most of length 64 bits.
So $xy = x * 2^{64} + y$ reciprocally for wv.
Now we will analyse each little multiplications we have to perform :
(x_ * w_) = (x * 2^{64} * w * 2^{64}) = (x * w * 2^{128})
(x_ * v) = (x * 2^{64} * v)
(y * w_) = y * w * 2^{64}

(y*v)
Going back to our code we have the variables a and b equal to xy and wv.

```
And with those lines we have
    a0.t[0] = a.t[0]; // a0 = y
    a1.t[0] = a.t[1]; // a1 = x

    b0.t[0] = b.t[0]; // b0 = v
    b1.t[0] = b.t[1]; // b1 = w
```

```
And with those one we start the multiplication
    //for this one we know that 2^128 modulo 2^115 -85 is equal to 696320
    a1b1 = a1.s * b1.s * 696320; // (x * w * 2^128)
    a0b1 = a0.s * b1.s; // y * w * 2^64
    a1b0 = a1.s * b0.s; // (x * 2^64 * v)
    a0b0 = a0.s * b0.s; // (y*v)
```

Now we have to apply the modulo for each multiplication and this line is doing this :

```
    a0b0 = ((a0b0 >> 115) * 85) + (a0b0 & m115.s);
```

And here is our explanation. Our idea is the following, if we want to do i [u-p] we can say that i = k(u-p) + r with k in Z. So i = ku - kp + r and if $kp < u$ we have i = r - kp [u] and we want to show that's what do the line of code above.
In our case the u is equal to 2^{115}, p is equal to 85 and i is the content of a0b0 that is on 128 bits.
If we perform a0b0 modulo 2^{115} it's the same as $a0b0\&(2^{115} - 1)$ which is equal to our number m115. So we have the value of our r for the formula above.
Now we want to get the kp. So going back to our operation a0b0 modulo 2^{115} the k is equal to a0b0 shift to the left by 115 so in C it's $a0b0 >> 115$. And now we have to multiply it by our p value corresponding to 85.

```
So the line :
    a0b0 = ((a0b0 >> 115) * 85) + (a0b0 & m115.s);
perform the modulo over 2^115 -85 for a number of size 2^128.
```

Now some other information. Each of our multiplication are strictly inferior to 2^{128}. Because the number we are multiplying are inferior to 2^{64}. And for the rest of the code here is what happened :

```
    mid.s = a0b1 + a1b0; // y * w * 2\^64 + (x * 2\^64 * v)
    mid_q.t[0] = mid.t[1]; //we take the first part of the number above
    // So mid_q = j * 2^128 So as the first operation it's equal to
    // j * 696320
    mid_q.s   *= 696320;

    //Now we do the rest of our addition
    mid_r.t[1] = mid.t[0];
    mid_r.s    = ((mid_r.s >> 115) * 85) + (mid_r.s & m115.s);

    res.s = a1b1 + a0b0 + mid_q.s + mid_r.s;
    res.s = (res.s >> 115) * 85 + (res.s & m115.s);

    res.s = res.s > mod.s ? res.s - mod.s : res.s;
    return res;
```

5 Question 4

$W = 2^{64}$ is a parameter given in the statement. From it, we can use the result given by the heuristic analysis and compute $p = log(W)/sqrt(W) = 64/2^{32} = 1/2^{26}$, $k = log(W)/2 = 32$ and $\mu = sqrt(W)/2 = 2^{31}$.
For the e_j values, it is recommended in the book to use 2^j values. So to obtain an average of 2^{31} we get the following values :

```
    ej[31]
    for i in range 0,31:
        ej[i] = 2^(i+4)
```

For the subsets $S_{1,..,k}$ we do a modulo 32 and we map 0 to subset 0, 1 to 1 etc...
As for the distinguishable values, since $p = 1/(2^{26})$, we do a modulo 2^{26} of the current value and if it is equal to zero, it is a distinguishable one.

6 Question 5

Our program works for the value $g^{2^{31}}$ but for the value given in the subject our algorithm do not find a distinguishable point where the 2 kangoroo's stop. It's probably because the e_j values we have choosen are not suitable in this example. Our function dlog64 works like this :

```
//We have 2 hash table one for the tame kangaroo and one for the wild
//gExponentEJ is a table which contains all values of g^ej
//ej is a table which contains all values of ej
maskForModulo = (1<<26) - 1 //Mask for
maskForEj = (1<<5) - 1; //Mask for modulo 32
wildKangaroo = target
tameKangaroo = g^(2^63)
index = 0
tameExponent = 2^63
wildExponent = 0
trapExponent = 0
result = 0

while(true)
    index  =  tameKangaroo.s & maskForEj
    tameKangaroo = mul11585(tameKangaroo, gExponentEJ[index])
    tameExponent += ej[index]
    if (tameKangaroo.s & maskForModulo) == 0
        //We have a distinguishable element
        //The function searchTameKangarooTable search in the hash table of the
        //tame kangaroo if the wild kangaroo has put a trap for his current value
        //And the second argument trapExponent return the exponent value of the trap
        if searchTameKangarooTable(tameKangaroo.s, &trapExponent)
            if trapExponent > tameExponent
                result.s = trapExponent - tabeExponent
            else
                result.s = tameExponent - trapExponent
            break
        else
            //We put a trap for the wild kangaroo
            insertWildKangarooTable(tame.s, tameExponent)

    index  =  wildKangaroo.s & maskForEj
    wildKangaroo = mul11585(wildKangaroo, gExponentEJ[index])
    wildExponent += ej[index]
    if (wildKangaroo.s & maskForModulo) == 0
        //We have a distinguishable element
        //The function searchWildKangarooTable search in the hash table of the
        //wild kangaroo if the tame kangaroo has put a trap for his current value
        //And the second argument trapExponent return the exponent value of the trap
        if searchWildKangarooTable(wildKangaroo.s, &trapExponent)
            if trapExponent > wildExponent
                result.s = trapExponent - wildExponent
            else
                result.s = wildExponent - trapExponent
            break
        else
            //We put a trap for the wild kangaroo
            insertTameKangarooTable(tame.s, tameExponent)
return result
```

7 Question 6

Depending on the value we choose for the e_j's values our implementation respects the heuristic or not. Depending on the value, the algorithm takes more or less time.

8 Question 7

If we change the value of the e_j's values the algorithm can take more or less time. Some values are more suitable to some problems. By changing the position of the starting point we have the same problem.