3C TIC *(Edición 21 Vol.6 – Nº 2*
Junio – Septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

Recepción: 10 de mayo de 2017
Aceptación: 13 de junio de 2017
Publicación: 29 de junio de 2017

ALGORITMO DE BOOTH EN OPERACIONES DE EXPONENCIACIÓN MODULAR

BOOTH ALGORITHM IN MODULAR EXPONENTIATION OPERATIONS

Jesús Ayuso Pérez[1]

1. Compositor musical y desarrollador. Licenciado en Ingeniería Informática por la Universidad Carlos III de Madrid (UC3M). E-Mail: ayusoperez@terra.com

Citación sugerida:

Ayuso Pérez, J. (2017). Algoritmo de Booth en operaciones de exponenciación modular. *3C TIC: Cuadernos de desarrollo aplicados a las TIC*, 6(2), 1-12. DOI: <http://dx.doi.org/10.17993/3ctic.2016.56.1-12/>.

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

RESUMEN

El algoritmo dado por Andrew Donald Booth en 1950 (Booth, 1951), presenta un comportamiento más natural de cara al cruce de operaciones y sucesiones de las mismas, cuando éstas se encuentran en el mismo estrato algebraico (Ayuso 2015, pp. 113-119). De ahí que el presente documento proponga varios algoritmos de exponenciación entre enteros basados en el concepto ideado por Booth. Mostrando distintas aportaciones para realizar ese cómputo, y sobre todo con la novedad de apoyarse en operaciones que igualmente ya explotan el concepto introducido por Booth, solucionando así el proceso de reducción en un contexto modular. Y jugando con el hecho de advertir que, para lograr el resultado final, se requerirán una gran cantidad de cálculos los cuales explotan estos conceptos.

ABSTRACT

The algorithm given by Andrew Donald Booth in 1950 (Booth, 1951), presents a more natural behavior to the crossing of operations and successions of the same, when these are in the same algebraic stratum (Ayuso 2015, pp. 113-119). Hence in this paper, we propose several algorithms of exponentiation between integers based on the concept devised by Booth. We will see different contributions to realize this computation, and above all with the novelty of supporting us in operations that equally already exploit the concept introduced by Booth, thus solving the reduction process in a modular context. And playing with the fact of warning that to achieve the final result, will require a lot of calculations which exploit these concepts.

PALABRAS CLAVE

Booth, Algoritmo, Adición, Exponenciación, Modular.

KEYWORDS

Booth, Algorithm, Addition, Exponentiation, Modular.

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

1. INTRODUCCIÓN

El concepto de Booth aplicado dentro de una aritmética modular permite ir reduciendo el resultado a la par que se van calculando las operaciones primitivas (Ayuso 2015, pp. 222-229), tales como las de adición o sustracción, de manera que al término de las mismas, obtenemos el resultado ya reducido, ahorrándonos así posteriores cómputos para encuadrar el elemento dentro del módulo en que estamos trabajando. Este tipo de cálculos son muy utilizados en distintos campos y especialmente en el mundo de la Criptografía (Diffie, W., and E. Hellmen 1976, pp. 644–654).

Partiendo de lo anterior, la operación de exponenciación es planteada como una sucesión de multiplicaciones y/o divisiones, entendidas estas últimas, al estar en un contexto modular, como multiplicaciones por el inverso modular, en las que la modularidad vendrá facilitada, por apoyarse en las implementaciones de primitivas modulares referenciadas: multiplicación, inverso, adición, sustracción... (Ayuso 2015/2016, pp.222-229/255-221/28-41) Concretamente, se detallarán los algoritmos usados para la operación de multiplicación modular, ya que serán adaptados un poco, buscando una consonancia perfecta con la operación que se busca construir: la de exponenciación modular. E igualmente, como se adelantaba, al tratar la división como una multiplicación por el inverso modular, también se detallará la implementación utilizada para el cálculo del inverso multiplicativo de un entero.

2. METODOLOGÍA Y RESULTADOS

En este apartado, lo primero que se hará será dar la implementación de multiplicación modular sobre la que se basará el presente artículo. Más concretamente, se procede a dar una versión partiendo de las implementaciones que se describen en la referencia bibliográfica titulada: '*Booth algorithm modular arithmetic operations of multiplication*' (Ayuso 2015, pp. 255-221).

Se tiene que la multiplicación de dos números, *a* y *b*, módulo *m*, los 3 de longitud *n*, sería:

```
result = 0;      //    ELEMENTO    NEUTRO    RESPECTO    OPERACION
ALGEBRAICA

opBooth = (0 0);// NINGUNA ACCION AL INICIO

weight = a;


for(int i = 0; i < n; i++)   {

     if(b[i] == 1) {

     switch(opBooth) {

           case ( 1 0 ):
```

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
                opBooth = mSubtraction1(weight, result);
                break;
          case ( 0 1 ):
          default:
                opBooth = mAddition1(result, weight, m);
                break;
          }
     }
     weight = mAddition2(weight, weight, m);
}
return result;
```

Figura 1: Algoritmo de multiplicación modular 1 (mMultiplication1).
Fuente: elaboración propia.

Puede observarse que la anterior implementación es exactamente igual que la que se expone en el trabajo '*Booth algorithm modular arithmetic operations of multiplication*' (Ayuso 2015, pp. 255-221) con la salvedad de que, comparados bien las 2, se aprecia que en la presente alternativa, cuando el resultado final no es el elemento correcto, sino el congruente del mismo, en el módulo en cuestión, no es realizado el último cálculo para obtener el resultado real; sino que éste es dejado tal cual ha sido obtenido. Más adelante queda explicado al detalle el porqué de esta decisión. También puede verse que ha sido etiquetada la operación (al pie del código), con el nombre de *mMultiplication1*, merece la pena destacarlo porque esa implementación será referida a con ese nombre, en lo que resta de documento.

Como se ha adelantado, la presente operación de división va a ser concebida como una operación de multiplicación por el inverso modular. Por ello, para construir nuestra versión del algoritmo de Booth para el cálculo de la exponenciación modular, se requerirá de la capacidad de poder obtener el inverso modular de un número *a* módulo *m*. Para ello, es utilizada la versión binaria del algoritmo de Euclides extendido (Euclid 1557). Dicho método está de sobra documentado, y figura como referencia en la bibliografía, el trabajo de Euclides donde se describe. A pesar de ello, a continuación se hace constar el pseudocódigo del algoritmo utilizado, ya que igualmente en su implementación se hace uso de primitivas que explotan el concepto de Booth, además de retocarlo también un poco. Tendríamos que el cálculo del inverso modular del número, *a*, módulo *m*, los 2 de longitud *n*, sería:

```
r0 = m
r1 = a
t0 = 0
t1 = 1
opBooth1 = (0 0);     // NINGUNA ACCION AL INICIO
opBooth2 = (0 0);     // NINGUNA ACCION AL INICIO
```

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
while(r1 != 0) {
 if(r0[0] = 0) {
      r0 >>= 1; // lo desplazamos a la derecha 1 bit
      if(t0[0] = 1)
      t0 = addition1(t0, m);
       t0 >>= 1; // lo desplazamos a la derecha 1 bit
 } else if(r1[0] = 0) {
      r1 >>= 1; // lo desplazamos a la derecha 1 bit
      if(t1[0] = 1)
      t1 = addition1(t1, m);
      t1 >>= 1; // lo desplazamos a la derecha 1 bit
 } else {
      if(r0 > r1) {
      r0 = subtraction1(r0, r1);
            if(opBooth2 == ( 1 0 )) {
                  if(opBooth1 == ( 1 0 )) {
                        switch(mSubtraction3(t0, t1)) {
                              case ( 1 0 ):
                                    opBooth1 = ( 0 1 );
                                    break;
                              default:
                                    break;
                        }
                  } else
                        opBooth1 = mAddition3(t0, t1, m);
            } else if(opBooth1 == ( 1 0 )) {
                        switch(mAddition3(t0, t1, m)) {
                              case ( 1 0 ):
                                    opBooth1 = ( 0 1 );
                                    break;
                              case ( 0 1 ):
```

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
                              opBooth1 = ( 1 0 );
                              break;
                        default:
                              break;
                  }
      } else
      opBooth1 = mSubtraction3(t0, t1);
} else {
r1 = subtraction1(r1, r0);
      if(opBooth1 == ( 1 0 )) {
            if(opBooth2 == ( 1 0 )) {
            switch(mSubtraction3(t1, t0)) {
                        case ( 1 0 ):
                              opBooth2 = ( 0 1 )
                              break;
                        default:
                              break;
                  }
            } else
                  opBooth2 = mAddition3(t1, t0, m);
      } else if(opBooth2 == ( 1 0 )) {
                  switch(mAddition3(t1, t0, m)) {
                        case ( 1 0 ):
                              opBooth2 = ( 0 1 );
                              break;
                        case ( 0 1 ):
                              opBooth2 = ( 1 0 );
                              break;
                        default:
                              break;
                  }
      } else
```

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
            opBooth2 = mSubtraction3(t1, t0)
        }
    }
}
return opBooth1;
```

Figura 2: Algoritmo de Euclides Extentido versión binaria (extendedEuclideanBinary).
Fuente: elaboración propia.

Tirando de bibliografía, es comprobable que la implementación anterior es exactamente igual que la que se expone en el trabajo '*Algoritmo de Booth en operaciones de Inverso Modular*' (Ayuso 2016, pp. 28-41) con la salvedad de que, al término del algoritmo, no se comprueba que el valor contenido en *opBooth1* sea una acción de Booth que indique que el resultado es el elemento correcto, en lugar de su inverso (aditivo, en este caso). Se toma la misma política que antes, al igual que con el algoritmo de multiplicación modular; de nuevo, insistir que más adelante se verá el porqué de estas decisiones.

En este punto, destacar que, como ha podido deducirse, que todas las versiones de los algoritmos utilizados retornan un valor que hace referencia a una acción de Booth, por ello se entiende, o se toma por convenio, que dejan el resultado del cómputo en la primera variable que se le pase como parámetro en la llamada. Por simplicidad.

Ahora se repasa la tabla dada por Booth para reducir el número de operaciones necesarias, la cual se apoya en la propiedad invertible de la operación con la que se construye nuestro cálculo sobre cierta estructura algebraica:

Tabla 1. Acciones de Booth.

bit menos significativo	bit extra	Interpretación	Acción
0	**0**	intermedio cadena de 0s	ninguna
0	**1**	final cadena de 1s	operación
1	**0**	comienzo cadena de 1s	operación inversa / inverso misma operación
1	**1**	intermedio cadena de 1s	ninguna

Fuente: elaboración propia.

Partiendo de la tabla anterior, por fin se entra en la operación que nos ocupa: la exponenciación modular. Nuestro algoritmo de Booth aplicado a la exponenciación de dos enteros, *a* elevado a *b* módulo *m*, los 3 de longitud *n* quedaría:

3C TIC (Edición 21) Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
result = 1;      //    ELEMENTO    NEUTRO    RESPECTO    OPERACION
ALGEBRAICA
opBooth1 = (0 0);     // NINGUNA ACCION AL INICIO
opBooth2 = (0 0);     // NINGUNA ACCION AL INICIO
bitExtra = 0;
weight = a;

for(int i = 0; i < n; i++)  {
     switch(actionBooth(b[i], bitExtra) {
       case ( 0 1 ):
     switch(mMultiplication1(result, weight, m)) {
               case ( 1 0 ):
               if(opBooth2 == ( 1 0 ))
                    opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 1 0 ) : ( 0 1 );
               else
                    opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 0 1 ) : ( 1 0 );
               break;
          default:
               if(opBooth2 == ( 1 0 ))
                    opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 0 1 ) : ( 1 0 );
               break;
     }
     break;
        case ( 1 0 ):
     iweight = 0; // nueva variable para el inverso
     switch(extendedEuclideanBinary(iweight, weight, m)) {
          case ( 1 0 ):
            switch(mMultiplication1(result, iweight, m)) {
               case ( 1 0 ):
```

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
                    if(opBooth2 == ( 1 0 ))
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 0 1 ) : ( 1 0 );
                    else
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 1 0 ) : ( 0 1 );
                    break;
               default:
                    if(opBooth2 == ( 1 0 ))
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 1 0 ) : ( 0 1 );
                    else
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 0 1 ) : ( 1 0 );
                    break;
          }
          break;
     default:
          switch(mMultiplication1(result, iweight, m)) {
               case ( 1 0 ):
                    if(opBooth2 == ( 1 0 ))
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 1 0 ) : ( 0 1 );
                    else
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 0 1 ) : ( 1 0 );
                    break;
               default:
                    if(opBooth2 == ( 1 0 ))
                         opBooth1 = opBooth1 == ( 1 0 ) ?
                              ( 0 1 ) : ( 1 0 );
                    break;
```

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

```
            }
            break;
      }
      break;
        default:
      break;
      }
      opBooth2 = mMultiplication1(weight, weight, m);
      bitExtra = b[i];
}

if(opBooth1 = ( 1 0 ))
      result = subtraction1(m, result);

return result;
```

Figura 3. Algoritmo de exponenciación modular.
Fuente: elaboración propia.

Es importante el hecho que se está dando por sentado que las operaciones, salvo *subtraction1*, retornan una acción de Booth, tal y como se especifica en su implementación, y el resultado del cálculo es dejado en la primera de las variables que le llega por parámetro en la llamada.

También apostillar que el concepto de Booth requiere la invertibilidad de la operación, luego en el algoritmo dado se entiende que trabaja siempre sobre una estructura algebraica donde todo elemento es invertible, es decir, para el caso que nos ocupa, entendemos que estamos ante un grupo multiplicativo.

Es necesario tener clara la forma en la que se aplica el concepto de Booth en cada parte del algoritmo anterior, para su completa compresión. Si nos fijamos, el primer *switch*, aplica el concepto de Booth en el sentido más clásico: reduce el número de operaciones a realizar apoyándose en la operación inversa; para el caso que nos ocupa, si entendemos la exponenciación como una sucesión de multiplicaciones, conseguimos reducir el número de multiplicaciones a realizar, ayudándonos de operaciones de división. Por otro lado, los *switchs* anidados nos sirven para ayudarnos de elementos inversos, y nuevamente reducir el número de operaciones; de ahí que en la métodos de multiplicación e inverso que hemos dado, hayamos eliminado la última operación, ya que ésta nos supondría un coste añadido, pues vamos a hacer cientos o miles de operaciones de este tipo para obtener el resultado de

3C TIC (Edición 21) Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

la exponenciación, y en cambio el concepto de Booth nos permite trabajar con elementos inversos. De los cuales sabemos que, si operan en el mismo estrato algebraico que la operación que vamos a realizar, tendrá el efecto inverso, como si operáramos por la operación inversa. En el caso que no ocupa, simplemente hemos de tener constancia de este detalle, para ser conscientes de que estamos proyectados contra un inverso, para poder deshacer dicha proyección algebraica en busca del elemento correcto. Por ello, cabe destacar esa última comprobación al final del código, en caso de que al salir del bucle, hayamos terminado obteniendo en elemento inverso, en lugar del elemento resultado: obtenemos su valor real con una simple resta contra el módulo sobre el que estamos trabajando (por tratarse de un inverso aditivo). No entramos en detalle de la implementación de esa operación de resta, *subtraction1*, porque es exactamente igual a la que figura en la bibliografía.

Evidentemente, también es posible utilizar la versión que soluciona el peor caso del algoritmo de Booth, donde 0s y 1s van intercalados en relación de uno a uno. Además, en cuanto a utilizar la representación NAF de los operandos (Reitwiesner 1960, pp. 231-308), para reducir aún más el número de operaciones, podría ser de gran ayuda de cara a ser pasado en dicha representación el módulo en que estamos trabajando a las operaciones primitivas de adición y sustracción, ya que, el valor de módulo, evidentemente, permanece inmutable, luego podríamos sacar un gran incremento en el rendimiento en la exponenciación si previamente al bucle de exponenciación, calculamos la representación NAF del módulo y trabajamos con ella en las operaciones primitivas. Además, existe una versión del algoritmo de exponenciación que evita ese cálculo constante del inverso modular, haciendo un recorrido a la inversa de los bits del exponente; también podría ser un buena técnica con la que complementar los conceptos aquí expuestos.

3. CONCLUSIONES

Aplicar el algoritmo de Booth en métodos que requieren de sucesivas operaciones relacionadas algebraicamente, nos ofrece un gran repertorio de herramientas para ahorrar operaciones ayudándonos de su inversa, apoyarnos en elementos inverso evitando actuaciones o eliminar operaciones cruzadas. Además, esa naturalidad con la que se comporta el concepto ideado por Booth en las distintas estructuras algebraicas lo suele hacer perfectamente compatible con el resto de técnicas de optimización que existan para el procedimiento que estemos estudiando.

En conclusión, los conceptos propuestos por Booth, son extensibles a distintos ámbitos y aplicables a muchos métodos que acaben trabajando sobre una aritmética construida con operaciones invertibles, ofreciendo la posibilidad de reducir el número de operaciones necesarias para determinados cálculos sobre dicha aritmética. Además de resultar mucho más elegante, traduciéndose en algoritmos con una convergencia mucho más natural y rápida. Como posible futura línea de investigación, se pueden tratar de aplicar los conceptos descritos en la aritmética de Montgomery (Montgomery 1985, pp. 519–521), ya que ésta permite realizar la operación de exponenciación de una manera más eficiente. Por lo que sería una técnica complementaria a la presente.

3C TIC *(Edición 21)* Vol.6 – Nº 2
Junio – septiembre '17, 1 - 12
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529
DOI: http://dx.doi.org/10.17993/3ctic.2017.56.1-12

4. REFERENCIAS BIBLIOGRÁFICAS

Booth, A. D. (1945), "A method of calculating reciprocal spacings for X-ray reflections from a monoclinic crystal," J. Sci. Instr, Vol. 22, p. 74.

Booth, A. D. and Britten, K. H. V. (1947), "General Considerations in the Design of an Electronic Computer".

Booth, A. D. (1951), "A signed binary multiplication technique", Q.J. Mech. and Appl. Math. Vol 4, No.2, pp.236-240.

W. Reitwiesner, George (1960), "Binary Arithmetic", 231-308.

Diffie, W., and E. Hellmen (1976). New directions in cryptography. IEEE Transactions on Information Theory,22(6), 644–654.

Peter Montgomery (1985.). Modular Multiplication Without Trial Division, Math. Computation, vol. 44, pp. 519–521.

Ayuso, J., "Booth algorithm operations addition and subtraction", 3C TIC. Vol 4, No.2, 2015, pp. 113-119.

Ayuso, J. (2015), "Booth algorithm modular arithmetic operations of addition and subtraction", 3C TIC. Vol 4, No.3, pp. 222-229.

Euclid of Alexandria (1557), "Elements", T.L. Heath's.

Ayuso, J. (2015), "Booth algorithm modular arithmetic operations of multiplication", 3C TIC. Vol 4, No.4, pp. 255-221.

Ayuso, J. (2016), "Algoritmo de Booth en operaciones de inverso modular", 3C TIC. Vol 5, No.2, pp. 28-41.