3C TIC *(Edición núm. 15)* Vol.4 – Nº 4
*Diciembre '15 – marzo '16, 255 - 265*
*Área de Innovación y Desarrollo, S.L.*
**ISSN: 2254 – 6529**
**DOI: http://dx.doi.org/10.17993/3ctic.2015.44.255-221**

**Recepción:** 17 de septiembre de 2015
**Aceptación:** 26 de noviembre de 2015
**Publicación:** 22 de diciembre de 2015

# ALGORITMO DE BOOTH EN ARITMÉTICA MODULAR PARA OPERACIONES DE MULTIPLICACIÓN

## BOOTH ALGORITHM MODULAR ARITHMETIC OPERATIONS OF MULTIPLICATION

Jesús Ayuso Pérez[1]

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

3C TIC *(Edición núm. 15)* Vol.4 – Nº 4
*Diciembre '15 – marzo '16, 255 - 265*
*Área de Innovación y Desarrollo, S.L.*
**ISSN: 2254 – 6529**

## RESUMEN

El algoritmo dado por Andrew Donald Booth en 1950 para la multiplicación, no es únicamente aplicable a dicha operación en los términos que describió en el citado trabajo, se puede aplicar a esa misma operación dentro de un contexto modular. De ahí que en el presente documento, propongamos varios algoritmos de multiplicación de enteros basados en el concepto ideado por Booth, pero que difieren de la solución que él propuso. Veremos distintas aportaciones para realizar ese cómputo, y sobre todo con la novedad de apoyarnos en operaciones que igualmente ya explotan el concepto introducido por Booth.

## ABSTRACT

The algorithm given by Andrew Donald Booth in 1950 for multiplying, is not only applicable to that transaction on the terms described in the cited work, it can be applied to the same operation within a modular environment. Hence, in this paper, we propose several algorithms for integer multiplication based on the concept developed by Booth, but that differ from the solution he proposed. We will see different contributions to make that count, especially with the news operations also rely on and exploit the concept introduced by Booth.

## PALABRAS CLAVE

Booth; algoritmo; multiplicación; modular.

## KEYWORDS

Booth; algorithm; multiplication; modular.

3C TIC *(Edición núm. 15)* Vol.4 – Nº 4
*Diciembre '15 – marzo '16, 255 - 265*
*Área de Innovación y Desarrollo, S.L.*
**ISSN: 2254 – 6529**

## INTRODUCCIÓN

Como adelantábamos, es cierto que Booth ya definió un método de multiplicación basado en el concepto que a día de hoy lleva su nombre, pero los algoritmos de multiplicación que vamos a mostrar en el presente trabajo, aunque usan esa idea, lo hacen con una visión distinta a la suya, y por consiguiente dando con métodos de multiplicación totalmente diferentes. Además veremos que este estudio está principalmente enmarcado en un contexto modular, lo cual no implica que los métodos no sean *migrables* a otras aritméticas, pero sí que demuestra que se desligan completamente de la idea original, ya que nunca se definió para operaciones modulares.

Por otra parte, el otro aspecto diferenciativo de las nuevas soluciones que vamos a describir es que éstas se apoyan o hacen uso de operaciones que a su vez también están utilizando el concepto de Booth. Es decir, entre las referencias estudiadas para el presente artículo, se encuentran métodos de suma y resta implementadas mediante el algoritmo de Booth. Nosotros haremos uso de esos conceptos para construir la operación de multiplicación modular; no sólo haremos uso de ellas, sino que explotaremos la particularidad de basarse en ese concepto para obtener un mayor rendimiento, una solución más natural, más elegante y simple... que la que se consigue con otros elaborados métodos de multiplicación de enteros.

Partiendo de esto, tenemos que nuestra operación de multiplicación la plantearemos como una sucesión de sumas y/o restas, en las que la modularidad nos vendrá facilitada, nos será transparente, por apoyarnos en las implementaciones referenciadas de adición y sustracción modular. Detallaremos las implementaciones usadas, ya que vamos a personalizarlas un poco, buscando adaptarlas a nuestras necesidades, además de utilizar varias versiones, lo que nos permitirá mostrar y diferenciar mejor los conceptos que queremos explicar en este trabajo.

3C TIC *(Edición núm. 15)* Vol.4 – Nº 4
*Diciembre '15 – marzo '16, 255 - 265*
*Área de Innovación y Desarrollo, S.L.*
**ISSN: 2254 – 6529**

## MÉTODOS

En este apartado, lo primero que haremos será dar la implementación de suma modular y resta modular sobre las que nos vamos a basar. Más concretamente, como decíamos, para diferenciar mejor lo conceptos que se intententa exponer, vamos a dar 2 versiones para cada operación partiendo de las implementaciones que se describen en las referencias bibliográficas tituladas: '*Algoritmo de Booth en operaciones de Adición y Sustracción*' y '*Algoritmo de Booth en aritmética modular para operaciones de Adición y Sustracción*'.

Empezaremos con la adición, veremos una primera versión del algoritmo de suma modular basándonos en la bibliografía referenciada. Tendríamos que la suma de dos números, *a* y *b*, módulo *m*, los 3 de longitud *n*, sería:

```
result = a;
for(int i = 0; i < n; i++)  {
      if(b[i] == 1) result = successor(result, i);
      if(m[i] == 1) result = predecessor(result, i);
}

if(result[n + 1] == 1)  {
      for(int i = 0; i < n + 1; i++)
            result[i] = ~result[i];    // OP. DE NEGADO EN ANSI
C
      result = successor(result, 0);

      return BOOTH(0, 1); // OPERACION INVERSA
} else
      return BOOTH(1, 0);   // OPERACION
```

Algoritmo de suma modular 1 (mAddition1).

Como podemos observar, la anterior implementación es exactamente igual que la que se expone en el trabajo '*Algoritmo de Booth en aritmética modular para operaciones de Adición y Sustracción*' con la salvedad de que devolvemos una acción de Booth, dependiendo de si el cómputo final desborda el módulo sobre el que trabajamos o no. Ya que dependiendo de esto, el resultado será un número positivo, o será el congruente negativo del entero en cuestión. Podemos ver que hemos etiquetado la operación (al pie del código), con el nombre de *mAddition1*, lo destacamos porque nos referiremos a esa implementación con ese nombre, en lo que resta de documento.

3C TIC (Edición núm. 15) Vol.4 – Nº 4
Diciembre '15 – marzo '16, 255 - 265
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529

Pasamos ahora al segundo algoritmo de adición modular que utilizaremos; lo llamaremos *mAddition2*. La principal diferencia con el anterior, será que para esta versión siempre trabajaremos con enteros positivos, y nunca con los congruentes negativos de estos, de manera que no se devuelve ningún tipo valor con respecto a eso:

```
result = a;
congruent = a;
for(int i = 0; i < n; i++)  {
      if(b[i] == 1) {
            result = successor(result, i);
            congruent = successor(congruent, i);
      }
      if(m[i] == 1) congruent = predecessor(congruent, i);
}
return result >= m ? congruent : result;
```

Algoritmo de suma modular 2 (mAddition2).

Analizando el código, lo único que hace el algoritmo anterior es llevar 2 cómputos, uno con el resultado sin reducir y el otro reduciendo el cálculo. Y finalmente, comprueba si el resultado ha desbordado el módulo, *m*, en cuyo caso devuelve el resultado reducido, y en caso de no ser así, devuelven el resultado sin reducir.

Vayamos ahora a la sustracción, al caso de la resta de dos número, *a* y *b* módulo *m*, de longitud *n*. En esta primera versión, *mSubtraction1*, la modularidad la vamos a gestionar con una primera comprobación de si el sustraendo es mayor que el minuendo:

```
if(b > a) {
      result = b;
      for(int i = 0; i < n; i++)  {
      if(a[i] == 1) result = predecessor(result, i);
      }
      return BOOTH(0, 1); // OPERACION INVERSA
} else {
      result = a;
      for(int i = 0; i < n; i++)  {
            if(b[i] == 1) result = predecessor(result, i);
      }
```

3C TIC (Edición núm. 15) Vol.4 – Nº 4
Diciembre '15 – marzo '16, 255 - 265
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529

```
      return BOOTH(1, 0);   // OPERACION
}
```

Algoritmo de resta modular 1 (mSubtraction1)

Si nos fijamos, para esta implementación, el valor del módulo, *m*, no interviene en los cálculos. Como avanzábamos, el motivo es porque nosotros mismos gestionamos que el resultado se encuentre siempre dentro del conjunto de elementos sobre los que trabajamos con la comprobación de que, en caso de que el sustraendo sea mayor que el minuendo, intercambiamos el papel de los valores en la operación (el sustraendo pasa a ser el minuendo y viceversa), de manera que en tal caso devolvemos el congruente positivo del elemento negativo que genera nuestro cómputo. Y por supuesto, devolvemos la acción de Booth acorde con ese detalle; es decir, que ese elemento debe implicar una operación inversa: en el caso que nos ocupa, de sustracción aunque se trate de un valor positivo.

Bien, al igual que para la adición, en este artículo usaremos una segunda implementación de la operación de sustracción, a la que nos referiremos como *mSubtraction2*. Y de igual modo que hemos hecho para esa segunda versión de la adición, en esta segunda versión de la sustracción no trabajaremos con congruentes positivos o negativos de elementos negativos o positivos, respectivamente. Tendríamos:

```
if(b > a) {
      result = b;
      for(int i = 0; i < n; i++)  {
            if(a[i] == 1) result = predecessor(result, i);
            if(m[i] == 1) result = successor(result, i);
      }
} else {
      result = a;
      for(int i = 0; i < n; i++)
            if(b[i] == 1) result = predecessor(result, i);
}
return result;
```

Algoritmo de resta modular 2 (mSubtraction2).

Vemos, que la anterior implementación, también comprueba si el sustraendo es mayor que el minuendo, pero esta vez para saber si tiene que ir aumentando el módulo *m*. En otro caso, es una simple resta.

3C TIC *(Edición núm. 15)* Vol.4 – Nº 4
*Diciembre '15 – marzo '16, 255 - 265*
*Área de Innovación y Desarrollo, S.L.*
**ISSN: 2254 – 6529**

Llegados a este punto, destacar que, como hemos podido deducir, en las versiones etiquetadas con un *1*, en que hacemos uso de congruentes; esas implementaciones, decíamos, entendemos que dejan el resultado del cómputo en la variable llamada *result* y retornan un valor que hace referencia a una acción de Booth. En cambio, en las versiones etiquetadas con un *2*, damos por sentado que simplemente retornan el resultado del cómputo, es decir, la propia variable *result*.

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

| 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 |

Tabla de acciones de Booth.

Partiendo de la tabla anterior, vamos por fin a entrar en la operación que nos ocupa: la multiplicación modular. Nuestro algoritmo de Booth aplicado a la multiplicación de dos enteros, *a* y *b* módulo *m*, de longitud *n* quedaría:

```
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) {

      swtich(opBooth) {

            case ( 1 0 ):

                  opBooth = mSubtraction1(weight, result);

                  break;

                  case ( 0 1 ):

            default:

                  opBooth = mAddition1(result, weight, m);

                  break;
```

3C TIC (Edición núm. 15) Vol.4 – Nº 4
Diciembre '15 – marzo '16, 255 - 265
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529

```
      }
      }
      weight = mAddition2(weight, weight, m);
}

if(opBooth = ( 1 0 ))
      mSubtraction1(m, result);

return result;
```

Algoritmo de multiplicación modular 1.

Recordamos que en el caso de las operaciones etiquetadas como *1*, éstas entendemos que dejan el resultado del cómputo en la propia variable *result*. Llegados a este punto, ya podemos ver que efectivamente la solución expuesta difiere considerablemente del enfoque que Booth utilizó para su versión de multiplicación de enteros. Como podemos apreciar, en la implementación que hay sobre estas líneas, se utiliza el algoritmo de Booth para apoyarnos en los elementos congruentes que van generando las implementaciones de adición y sustracción utilizadas (además de para implementar las propias operaciones de suma y resta).

Cabe destacar esa última comprobación al final del código, en caso de que al salir del bucle, hayamos terminado obteniendo en elemento congruente, en lugar del elemento resultado: obtenermos su valor real con una simple resta contra el módulo sobre el que estamos trabajando.

Vayamos ahora a otro algoritmo de multiplicación de dos números, *a* y *b* módulo *m*, de longitud *n* sería:

```
result = 0;      //    ELEMENTO    NEUTRO    RESPECTO    OPERACION
ALGEBRAICA
bitExtra = 0;
weight = a;

for(int i = 0; i < n; i++)  {
      swtich(actionBooth(b[i], bitExtra) {
            case ( 0 1 ):
                  result = mAddition2(result, weight, m);
                  break;
```

3C TIC (Edición núm. 15) Vol.4 – Nº 4
Diciembre '15 – marzo '16, 255 - 265
Área de Innovación y Desarrollo, S.L.
ISSN: 2254 – 6529

```
            case ( 1 0 ):
                  result = mSubtraction2(result, weight, m);
                  break;
      }
      weight = mAddition2(weight, weight, m);
      bitExtra = b[i];
}
return result;
```

Algoritmo de multiplicación modular 2.

La anterior versión sí puede parecerse más a la idea original de Booth, con el matiz de que está adaptada a un contexto modular y que hace uso de operaciones (adicción y sustracción) que igualmente se apoyan en el propio algoritmo de Booth. También queríamos hacerla constar ya que es perfectamente susceptible de ser complementada con la otra implementación. Es decir, los conceptos son perfectamente compatibles entre sí, para explotar aún más el rendimiento del cómputo reduciendo el número de operaciones a realizar.

Evidentemente, también podríamos usar la versión que soluciona el peor caso del algoritmo de Booth, donde 0s y 1s van intercalados en relación de uno a uno. En cuanto a utilizar la representación NAF de los operandos, para reducir aún más el número de operaciones, podría sobrecarga demasiado una operación tan ligera como es la operación de multiplicación modular. Al igual que si intentáramos usarla para implementar la suma y/o resta modular.

3C TIC *(Edición núm. 15)* Vol.4 – Nº 4
*Diciembre '15 – marzo '16, 255 - 265*
*Área de Innovación y Desarrollo, S.L.*
**ISSN: 2254 – 6529**

## DISCUSIÓN/CONCLUSIÓN

El aplicar el algoritmo de Booth para trabajar con elementos congruentes dentro de un contexto modular, nos ofrece una alternativa más natural para el cálculo de operaciones de multiplicación, ya que nos ofrece la posibilidad de trabajar siempre con elementos enmarcados dentro del módulo en que estamos operando, sin apenas costes añadidos. Además de, por supuesto, reducir el número de operaciones necesarias para obtener el resultado final.

En conclusión, la idea propuesta por Booth, se adapta mejor a los cálculos modulares, ofreciendo una solución más flexible y acorde con las operaciones con las que se construye la estructura algebraica sobre la que estamos trabajando. Además de resultar mucho más simple, traduciéndose en mucho menos código o hardware para su implementación.

## BIBLIOGRAFÍA

Booth, A. D., "A method of calculating reciprocal spacings for X-ray reflections from a monoclinic crystal," J. Sci. Instr, Vol. 22, 1945, p. 74. http://dx.doi.org/10.1088/0950-7671/22/4/404

Burks, A., Goldstein, H. and Von Neumann, J., "Logical Design of an Electronic Computing Instrument" (Princeton, 1946).

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

Booth, A. D., "A signed binary multiplication technique", Q.J. Mech. and Appl. Math. Vol 4, No.2, 1951, pp.236-240. http://dx.doi.org/10.1093/qjmam/4.2.236

Ayuso, J., "Booth algorithm operations addition and subtraction", 3C TIC. Vol 4, No.2, 2015, pp.113–119. http://dx.doi.org/10.17993/3ctic.2015.42.113-119

Ayuso, J., "Algoritmo de Booth en aritmética modular para operaciones de Adición y Sustracción", 3C TIC. Vol 4, No.3, 2015, pp.222–229. http://dx.doi.org/10.17993/3ctic.2015.43.222-229