TÓPICOS SOBRE OPTIMIZACIÓN DE ALGORITMOS

Gloria Inés Giraldo Echeverry
Ingeniera industrial, Universidad Tecnológica de Pereira.
Magister en Ingeniería de Sistemas,
Universidad Nacional de Colombia.
Profesora asociada Departamento de Ingeniería de Sistemas
Universidad Nacional de Colombia, Bogotá.

Introducción

El presente artículo pretende ilustrar cómo, en el desarrollo de sofware, un primer algoritmo que se construya en la solución de un problema puede mejorarse muchísimo. analizando dónde se gasta su tiempo.

En el artículo publicado en la revista *Ingeniería e Investigación* 32 por la autora y titulado "La eficiencia de los algoritmos" [4] se explicó en que consiste la eficiencia de un algoritmo y cómo realizar el cálculo de complejidad cuya base es la notación-0. Estos aspectos constituyen el elemento de análisis básico que nos permite aproximar el desempeño del algoritmo y, por tanto, no se detallarán aquí.

La mejora de un algoritmo puede lograrse de diversas maneras; eso depende de las características del mismo.

Una técnica aplicada con frecuencia es la llamada dividir y conquistar, que consiste en dividir el problema en subproblemas, de tal forma que, combinando las soluciones de los subproblemas, se construya la solución del problema original de una manera fácil. Está técnica hace que los algoritmos sean más eficientes, especialmente cuando a través de ella se logra bajar de complejidad.

En ocasiones se logra una mejora sustancial del algoritmo, si se cambian las reglas de base. En este caso, la escogencia adecuada de las estructuras de datos, por ejemplo árboles en lugar de arreglos, puede convertirse en una importante herramienta para construir algoritmos eficientes.

Otras veces se logra mejorar un algoritmo cambiando el algoritmo mismo: por ejemplo. se escogió como estructura un árbol pero se cambia la forma de construirlo y/o la forma de recorrerlo.

La optimización de un algoritmo se obtiene cuando se hace la escogencia correcta de algoritmo y estructura de datos.

Para ilustrar como puede lograrse esto, se escogió la multiplicación de matrices booleanas, por ser uno de los algoritmos en que se ha logrado mejoras significativas.

MATRICES BOOLEANAS

La primera opción para multiplicar matrices booleanas es usar el algoritmo de multiplicación tradicional, basado en multiplicar una fila por una columna para ir determinando cada término. Sin embargo, ésta es una opción altamente ineficiente por el alto grado de complejidad.

Posteriormente se explica un algoritmo basado en el concepto de bits paralelos, con el fin de hacer más fácilmente entendible la aplicación de este concepto en el algoritmo Four-Russians.

Finalmente, se explica el algoritmo que hoy día se considera el óptimo para multiplicación de matrices booleanas que es el llamado algoritmo Four-Russians o algoritmo de los cuatros rusos que logró incorporar el concepto de bits paralelos y la aplicación de la técnica dividir y conquistar.

A. Algorimo de multiplicacidn tradicional

Las operaciónes booleanas son:

AND

.	0	1
0	0	0
1	0	1

OR

+	0	1
0	0	1
1	1	1

```
Program TRADICIONAL;
(*calcula el producto de una matriz A, por una *)
(*matriz B en una matriz C*)
Var     i, j, k: integer;
        suma: integer;
        A: array [1.. m, 1.. n] of integer;
        B: array [1.. n, 1.. p] of integer;
        C: array [1..m, 1..p] of integer;

begin
        for i: = 1 to m do
            for j:= 1 to p do
                begin
                    suma : = 0;
                    for k : = 1 to n do
                        suma : = suma +A [i,k]*B[k,j];
                        C[i,j] := suma
                end
end
```

La complejidad de este algoritmo se determina aplicando la propiedad multiplicativa de la notación - O [4], lo que equivaldria a $O(m*p*n)$. En general, para multiplicar una matriz A de $n*n$ elementos, por una matriz B de n*n elementos la complejidad es O (n^3).

B. Concepto de operaciones de bits paralelos

El concepto de operaciones de bits paralelos para multiplicar matrices booleanas se explica a continuación.

Si se tienen las matrices A(u*v) y B(v*w) es posible obtener la multiplicación de las matrices en C uniendo varias filas de B [Smith, 1987].

Sean las matrices:

$$\begin{vmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{vmatrix} \quad * \quad \begin{vmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{vmatrix} \quad = \quad \begin{vmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{vmatrix}$$

A(u*v) B(v*w) C(u*w)

Se toman las filas de la matriz A una por una por ejemplo, la primera 0 1 0 1, puesto que están prendidos los bits de la posición 2 y 4 se toman estas filas de B y se suman para dar la primera fila de C.

Primera fila de A Filas 2 y 4 de B

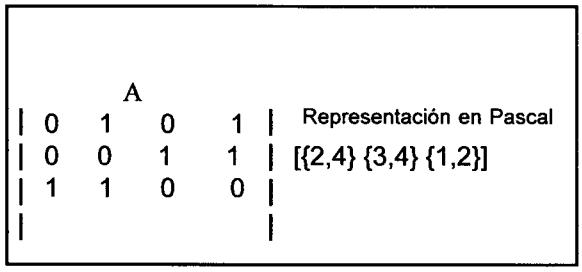
0101 01001

+10001

11001 → Primera fila de C

El concepto de bits paralelos será utilizado en el algoritmo BITS PARALELOS con el fin de hacer más fácilmente entendible el algoritmo posteriormente presentado con el nombre de Four-Russians.

En el algoritmo en Pascal i:=iésimo valor lógico, correspondiente a la presencia o ausencia de i en el conjunto, las matrices serán representadas por un vector de conjuntos de la siguiente manera:



Lo anterior lleva a que cada matriz se almacene en una arreglo de una sola dimensión; que contiene tantos conjuntos, como filas tenga la matriz.

Está representación permite utilizar los operadores que Pascal dispone para actuar sobre conjuntos, de modo análogo a las operaciones de teoría de conjuntos en matemáticas: unión equivale a + y pertenencia a IN; utilizadas en el algoritmo.

```
Program BITS - PARALELOS

(*multiplicación de matrices booleanas C= A*B *)
const u = 3 (* número de filas en A y C *)
      v = 4 (* número de columnas en A y filas en B*)
      w = 5 (* número de columnas en B y C *)
type  setv = set of 1.. v;
      setw = set of 1.. w;
var   i, k: integer;
      A: array [1..u] of setv;
      B: array [1..v] of setw:
      C: array [1..u] of setw;
Begin
     for i: = l to u do
          begin
            C [i] : =[];
             for k: = 1 to v do
                 if k in A [i]
                   then C [i] := C[i] + B [k]
          end
end
```

C. Algoritmo Four-Russians

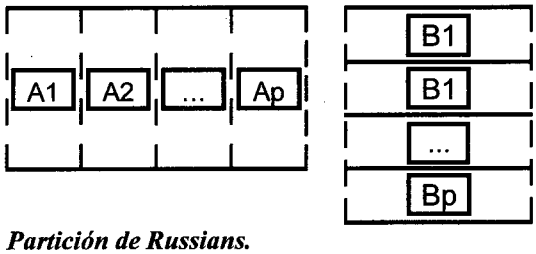
El algoritmo Four-Russians (Aho, 1974) está basado en la técnica de multiplicación de matrices por bloques.

Para un mejor entendimiento de algoritmo se ilustrará con dos matrices cuadradas A y B de dimensiones *n*n.*

Cada matriz será partida en un total de p submatrices. Cada submatriz se referenciara por h, donde *h = 1,2...p.* Cada submatriz A_h contendrá columnas y cada submatriz B_h contendra n filas.

Se definen: $m = lg_2 n$ y $p = n/m$, tomando la parte entera para ambas relaciones.

A es partida en p submatrices A_h y B es partida en *p* submatrices B_h. Si las matrices no se pueden dividir igualmente, es necesario rellenar A con columnas extras de 0 y B con filas extras de 0, de tal forma que las matrices queden cuadradas y del mismo tamaño (ver gráfica de partición.).



Partición de Russians.

Como resultado de la partición, el producto de las matrices C = A*B, es igual a la sumatoria de A_h*B_h, donde cada producto A_h*B_h es una matriz n*n.

Sea A [h,i] la i-ésima fila de la matriz h y B [h,k] la k-ésima fila de la matriz h.

Sin embargo, hay m elementos en cualquier fila de una submatriz A_h; así que puede haber solamente $q=2^m$ filas distintas en una submatriz A_h. La base del algoritmo Russians es que para cada $1 \leq h \leq p$, se calculan y almacenan en una tabla las *q* posibles combinaciones necesarias de las filas de B_h, y se usa el valor de cada fila de A_h como un subíndice de está tabla. Esto es ilustrado en el arreglo BCOMB.

Consecuentemente, cada fila A [h,i] es interpretada de izquierda a derecha como un entero para indexamiento en BCOMB.

Para el algoritmo en Pascal, cada matriz está representada por un arreglo en dos dimensiones de la siguiente manera: cada submatriz es un vector de conjuntos, representado de la misma forma que en el algoritmo de bits paralelos, asi que una submatriz queda completamente almacenada en una fila, y puesto que son p submatrices habra este número de filas tanto para la matriz A, como para la matriz B. El número de columnas para la matriz A es n y para la matriz B es m correspondientes a la cantidad de conjuntos que tiene cada submatriz, recordando que cada conjunto equivale a una fila de la submatriz.

```
Program RUSSIANS;

const   m = 3;
( *   lg2 n, parte entera *)
        n = 12; (* orden de las matrices cuadrada A,B,C *)
        p = 4; (* factor de partición n/m, parte entera*)
        q = 7; (* (2^m)- 1, para utilizar la posición 0 *)
type    setm = set of 1..m;
        setn = set of 1..n;

var     h, i, k, u: integer;
        A: array [1..p,1..n] of setm;
        B: array [1..p,1..m] of setn;
        C: array [1..n]      of setn;
        BCOMB: array [0..q] of setn;

function CONVERTIR (s: setm): integer;
        var i, j: integer;
        begin
                i:=0;
                for j :=m downto 1 do
                   i := 2 * i+ord(j in s);
                CONVERTIR:=i
        end;
begin
    for i:= 1 to n do
        C [i]: = [];
    for h:= 1 to p do
        begin
            BCOMB [0]: = [];
                    j: = 0;
                    k: =1;
                    u: = l;
                    for i:= 1 to q do
                       begin (*genera BCOMB desde B*)
                                BCOMB [i]:=BCOMB [j]+B[h,k];
                                j: =j+1
                                if j = u
                                   then begin
                                                j: = 0;
                                                k: = k + l;
                                                u: =i +1
                                        end
                       end
                    for i:=l to n do
                          (*indexar BCOMB por A y aplicar a C *)
                        C[ i] : = C [ i] + BCOMB [ CONVERTIR (A[h, i ] ) ]
        end
end
```

Para ilustrar el algoritmo, considérense las matrices A y B mostradas a continuación. Para este caso se tiene n=12, m=3, p=4. Para las submatrices A_1*B_1; se calcula el valor de BCOMB y los valores de C. Es de aclarar que para obtener el resultado final del algoritmo habría que continuar los cálculos de A_2*B_2, A_3*B_3, A_4*B_4.

000	000	001	001
110	011	001	100
000	000	001	100
110	000	000	100
100	100	011	000
000	001	010	000
100	100	000	000
000	000	010	000
001	000	000	000
100	101	110	111
100	000	010	000
100	001	000	000

Matriz A

0	0	0	0	0	1	1	0	1	0	0	0
1	1	0	0	1	1	0	0	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	1	0	0
0	1	0	0	0	0	1	0	0	0	0	0
0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1

Matriz B

Seguimiento del algoritmo Russians

Para h = 1 $A_1* B_1= C_1$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	1	0	0	0
1	1	0	0	1	1	0	0	1	0	0	0
1	1	0	0	1	1	1	0	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	1
0	1	1	0	1	1	1	0	1	0	0	1
1	1	1	0	1	1	0	0	1	0	0	1
1	1	1	0	1	1	1	0	1	0	0	1

BCOMB

0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	0	1	0	0	0
0	0	0	0	0	1	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0	1	0	0	0
0	0	0	0	0	1	1	0	1	0	0	0
0	0	0	0	0	1	1	0	1	0	0	0

C

¿Cúal es la complejidad del algoritmo RUSSIANS?.

Si se analiza el trabajo desarrollado por cada paso es:

Máx (n, p*q, p*n*m); lo que equivale a O (m*n*p), teniendo en cuenta que p=n/m, la complejidad sería O (n^2).
La pregunta sería: ¿por qué no mejoró la complejidad del algoritmo como era de esperarse? Eso depende de su implementación.

El trabajo desarrollado por CONVERTIR tiene O (m), si se logra desarrollar esta función en O (1) [Smith, 1987] la complejidad del algoritmo sería:
O (np) = O ($n^2/lg_2 n$)

Como se anotó mejorará la complejidad de la función CONVERTIR. Para esto es necesario tener la facilidad de convertir entre primitivas (boolean, caracter, integer, real). La conversión de tipos puede lograrse aplicando formatos alternos.

El programa Russians requiere la habilidad de tomar el valor de un conjunto variable y convertirlo a entero.

Para lograr hacer la conversión inmediata se mapea el conjunto de elementos a...b a los 64 bits de una doble palabra.

|63| 62 |...|32| 31|...| 1| 0|

Mapeando la i-ésima columna de una palabra sobre el i-ésimo elemento de un conjunto. Con esta implementación, los elementos del conjunto aparecen en orden inverso al de la doble palabra, conduciendo a la función CONVERTIR 1.

Arbitrariamente se construyó un tipo enumerativo con dos valores mnemónicos bit e int. La forma se refiere a un conjunto de 64 bits representado por b, aunque luego se refiere a dos enteros de 32 bits cada uno representados por i1 e i2. Sin embargo, suponemos que el orden más alto del entero i1 es siempre 0. Puesto que los conjuntos en Russians son definidos en términos de 0...m, debemos usar la división entera div para descartar el menor bit significativo.

```
function CONVERTIR1 (s: setm): integer;
            type flag = (bit, int);
             setm = set of 1 .. m;
    var      vista: record case flag of
                        bit: (b: setm);
                         int: (il, i2: integer)
                    end;
begin
        vista.b: = s;
        CONVERTIR 1: = vista. i2 div 2
end;
```

CONCLUSIONES

La optimización de algoritmos hace referencia al análisis cuidadoso de su desempeño para analizar las fallas y concebir mejoras antes de llevarlos al computador.

Lo usual es que un primer algoritmo que se nos ocurra puede mejorarse muchísimo. La exposición sobre las distintas formas de multiplicar las matrices booleanas tiene por objeto ilustrar algunos de los aspectos referentes a la optimización de algoritmos.

BIBLIOGRAFIA

1. AHO, A.V., HOPCRPFT, J.E, and ULLMAN, J.D. *"The design and Analysis of Computer Algorithms"*. Addisson Wesley Publishing Company, E.U.A. 1974 pp.242-247.

2. ___*Foundations of Computer Science.* Computer Science Press, New York, E.U.A.1995 pp. 89-108.

3. BOHÓRQUEZ, J. *Análisis de algoritmos.* Versión preliminar, Facultad de Ingeniería, Universidad de los Andes, Bogotá Colombia 1990. pp. 92-104.

4. GIRALDO, G. "La eficiencia de los algoritmos". *Ingeniería e Investigación.* Universidad Nacional de Colombia, Facultad de Ingeniería, vol. 32, pp. 47-50.

5. KONVALINA, J. and WILEMAN, S. *Programming with Pascal.* De. Mc Graw Hill Inc. E.U.A. 1987.

6. SMITH, H.F. *Data Structure.* Harcourt Brace Jovanovich Publishers, Orlando, E.U.A.pp. 62-66 1987.