UNA APLICACIÓN DE LA FUNCIÓN MODULO (fmod) EN LENGUAJE C

RESUMEN
Una aplicación importante de la función modulo (fmod) en el anillo de los números enteros, es la clasificación, almacenamiento de datos y la introducción a la teoría de códigos. El problema que se trata corresponde a la clasificación de códigos de seguridad de una empresa con el fin de optimizar el espacio de memoria.

PALABRAS CLAVES:
Congruencia, diofanticas, división Euclidea, fragmentar, memoria, modulo.

ABSTRACT
An important application of the module function (fmod) in the ring of integer number, is the classification, data storage and the introduction to the code teory. The problem that is being studied corresponds to the classification of security codes of company in order to optimize memory space.

KEYWORDS: congruence, diofánticas, división Euclidea fragments, memory, module

ORLANDO RODRÍGUEZ BUITRAGO
Esp Matemáticas Computacional.
Profesor Asistente.
Universidad del Cauca.
orbuitra@unicauca.edu.co

GUILLERMO SOLARTE MARTINEZ
Ingeniero de Sistemas
Profesor Auxiliar
Universidad Tecnológica de Pereira.
roberto@utp.edu.co

1. INTRODUCCIÓN

Este artículo trata conceptos aritméticos en el anillo de los números enteros Z y el anillo cociente por la relación de congruencia modulo m, Z_m.

En diversos problemas de clasificación se establecen relaciones que existen entre los sistemas de numeración y las congruencias, las ecuaciones diofánticas y las ecuaciones de congruencia.

Una aplicación de las relaciones de congruencia es la relación modulo m, la cual se aplicara en la clasificación de códigos de seguridad de los funcionarios de una empresa

La función modulo se define:

> Sea $R = \{(a,b) \in Z \times Z, m \in Z^+ \mid a \equiv b(\text{mod } m)\}$
> Es decir, $a \equiv b(\text{mod } m)$ si y sólo si a y b dan el mismo residuo al ser divididos entre m.

A continuación se enumeran algunos puntos que son prerrequisitos necesarios para la comprensión de este tema. Estos puntos son:

- Sistemas de numeración, es decir de a en base b, con a y b números enteros positivos, b>1.
- Propiedades de la divisibilidad en Z.
- Números primos.
- Congruencias modulo m
- Suma y productos en Z_m.

Fecha de Recepción: 01 Agosto de 2006
Fecha de Aceptación: 30 Octubre de 2006

2. CLASIFICACIÓN DE ARCHIVOS.

Función fragmentar

Un problema clásico, es el manejo de los códigos de seguridad de una empresa, donde se requiere disponer de un archivo para guardar la información de los empleados en un computador. Los datos referentes a cada empleado están dispuestos en un registro, donde el número de identificación o código de dicho registro es un número de 8 o más dígitos, los cuales ocupan mucho espacio en memoria.

En lugar de esto, se puede emplear un método sistemático para el almacenamiento que sea de fácil acceso. Este consiste en utilizar una función llamada función fragmentar o de desmenuzamiento, que trabaja con aritmética modular. Si se conoce el número de datos de entrada, por ejemplo m, se puede construir una función h que asigne a cada código un número de uno a m. Este número nos servirá de índice para seleccionar una posición en la tabla formada por los m registros.

La función fragmentar se define como una aplicación

$$\begin{aligned} h:\ & Z \rightarrow Z_m \\ & k \rightarrow h(k) \qquad \text{con } 0 \le h(k) < m \end{aligned}$$

Donde k es el código del archivo a almacenar y $m \in Z^+$. Se trata de construir formalmente una aplicación que vaya del conjunto de los códigos al conjunto de la direcciones de memoria; es decir, dado un valor de

código, la función h proporciona una dirección o localización del registro correspondiente.

La función proporciona dicha dirección ejecutando algún tipo sencillo de operación aritmética en el propio código sobre la parte del mismo; es por esto que las funciones de fragmentación más utilizadas son las funciones *modulo* (si los códigos son alfanuméricas es necesario un pretratamiento de los datos para transfórmalos en números), sin embargo, como el espacio de los valores de los código es generalmente mucho más grande que el de las direcciones, algunos valores tendrán asociado la misma dirección. Es decir, las funciones de fragmentación no son inyectivas.

En nuestro ejemplo k es un código de seguridad. Supongamos que se tiene que almacenar un total de 2000 códigos y se dispone de 5000 posiciones de memoria.

En lugar de guardar las 2000 códigos puede tomar entonces m = 5000, o bien un número primo próximo a 5000 por ejemplo m = 4969, y almacenar h(k), es decir k modulo m.

Como la función modulo (fmod) no es inyectiva, es bastante probable que dos códigos tengan asignada la misma imagen por h, cuando esto ocurre, es decir, existen k_1 y k_2 distintos tales que $h(k_1) = h(k_2)$ (tiene la misma posición de memoria), se dice que se ha producido una doble asignación con una misma dirección. Es necesario un método que corrija este problema para que la asignación de memoria sea única en dicho archivo.

Con el problema anterior surgen dos tipos de preguntas:

1. ¿Que tipo de funciones de fragmentación hay que utilizar?
2. ¿Como se resuelve el problema de la doble asignación?

2.1 Elección de la función de Fragmentación

La elección de la función de fragmentación h(k), que transforma códigos en posiciones en la tabla es uno de los aspectos más críticos. Las dos exigencias fundamentales para una función de fragmentación son:

Que sea fácil de calcular y que minimice las dobles asignaciones, distribuyendo los códigos en la tabla de la forma más homogénea posible.

Si la tabla tiene m entradas se puede tomar como la función de fragmentación h(k).

$$h: Z \rightarrow Z_m$$
$$k \rightarrow h(k) := k \text{ (modulo m)}$$

La elección de m es muy importante

- No es conveniente elegir como m un número par, ya que si m es divisible por dos, entonces k es par si y solo si, h(k) es par , lo cual esta en contra del criterio de que, la función de fragmentación debe generar índices lo más aleatoriamente posible.
- Tampoco es aconsejable que m sea potencia de 10, ya que si $m= 10^n$ y los códigos k están en base 10, provocaría esto simplemente un truncamiento en los dígitos de código.

Por ejemplo, para los códigos k_1=34.456.875, k_2 = 8.306.875 y k_3 = 516.865, las posiciones de memoria cuando la función de fragmentación es h(k)= k (modulo 1000), son respectivamente:

8	7	5

8	7	5

8	6	5

Es decir, números muy diferentes pueden ocupar en la memoria la misma posición o una muy parecida.

En este caso, hallar el resto de la división supone quedarse con los dígitos menos significativos del código, y códigos con el mismo sufijo se colocarían en la misma posición de la tabla. De hecho, si la tabla fragmentar se utiliza para guardar los identificadores de un programa, esta seria una mala elección de m, puesto que los programadores tienden a utilizar nombres de variables como W1, V1, MACA1,.., variables con el mismo sufijo. En general, se pueden dar argumentos para ver si m no es primo y si h(k) no genera índices de manera aleatoria.

2.2 Resolución del problema de la doble asignación

Se dice que un problema produce una doble asignación cuando existe k_1 y k_2, $k_1 \neq k_2$, tales que $h(k_1) = h(k_2)$ (tiene asignada la misma posición en memoria).

Una forma de resolver el problema es buscar una posición en memoria que no esté ocupada para almacenar el archivo que produce la doble asignación. Para ello se podría sugerir por ejemplo, comenzando con la función fragmentar original, una sucesión de puestos en memoria h_0, h_1, h_2,.... de tal manera que:

i) h (k) := h_0 (k) si ésta posición esta libre; en caso contrario
ii) h(k) := h_1(k)) si ésta posición esta libre,... y así sucesivamente.

Esta técnica se llama direccionamiento vacío. Se podría elegir como sucesión por
ejemplo:
h_j (k) = (h_0 (k) + j) (modulo m),
con $0 \leq h_j (k) < m$.

Dicho de otra forma, se busca sucesivamente en la posición siguiente, considerando la tabla de forma circular, hasta encontrar una posición vacía.
Esta función coloca al archivo con código k detrás de h(k) y tan cerca de él como sea posible. Además, con esta elección de $h_j(k)$ se comprueban todas las posiciones en memoria hasta que se consigue una libre. Este método recibe el nombre de inspección lineal.
Desafortunadamente esta elección de $h_j(k)$ presenta dificultades; los archivos doblemente asignados tienden a acumularse (to cluster). En efecto si $k_1 \neq k_2$ y $h_i(k_i) = h_j(k_2)$ para ciertos i, j > 0, entonces $h_{i+s}(k_1) = h_{j+s}(k_2)$, s = 1,2.....

Esto hace poco eficaz la anterior elección de $h_j(k)$, pues en este caso, se produce la misma sucesión de posiciones cuando hay una doble asignación. Interesan entonces, funciones que distribuyan del modo más uniforme los códigos doblemente asignados con poco costo computacional.
Para evitar la acumulación (clustering) se usan otras formas de direccionamiento. Una de ellas es la técnica llamada doble fragmentación, Escogiendo como antes

$h(k) \equiv k \pmod{m}$ $0 \leq h(k) < m$

con m primo, se considera luego como segunda función fragmentar $g(k) \equiv k+1 \pmod{m-2}$, $0 \leq g(k) < m-1$ y como sucesión:

$h_j(k) = h(k) + j.g(k) \pmod{m}$, $0 \leq h_j(k) < m$

Se puede probar que, con esta estrategia, todas las posiciones en memoria son comprobadas y además, el problema del cluster se minimiza.

Nota: En la doble defragmentación se produce clustering si:

$$\exists\ k_1, k_2\ (k_1 \neq k_2) \text{ y } \exists\ i,j \text{ tales que } \begin{cases} h_i(k_1) = h_j(k_2) \\ h_{i+1}(k_1) = h_{j+1}(k_2) \end{cases}$$

(es decir si coinciden dos términos consecutivos de las sucesiones asociadas a k_i y k_j). En este caso se puede probar que, m y m-2 son números primos, entonces $k_1 \equiv k_2 \pmod{m(m-2)}$. Luego, para que la sucesiones asociadas a los códigos k_1, k_2 coincidan en dos términos consecutivos , con m y m-2 primos, es necesario que $k_1 \equiv k_2 \pmod{m(m-2)}$, cosa que de hecho no puede ocurrir si m(m-2) > h(k), para todo código k.
Volviendo de nuevo al ejemplo de la introducción, supongamos que se quiere asignar posiciones en memoria para los diez empleados siguientes:

k_1= 70560069 k_2 = 49731278
k_3= 29556190 k_4 =33288857
k_5= 10191166 k_6 = 33739186
k_7= 16349561 k_8 = 5119758
k_9 =13660359 k_{10} = 28637873

En este caso m = 4969, en principio hay que calcular congruencias modulo (fmod) 4969. Con El programa C, el problema se resuelve simplificando las expresiones

x = fmod (k_1 , 4969) , y = fmod (k_2 , 4969)
z = fmod (k_3, 4969)

Que dán como resultado x =1526, y =1526 y
z = 578 respectivamente, y así se asigna a estos tres archivos dichas posiciones. Como $k_4 \equiv 1526 \pmod{4969}$ y $h(k_2)$ = 1526, posición que ya está ocupada. Se calcula:

$h_1(k_4) \equiv h(k_4) + g(k_4)$, (modulo 4969) con
$g(k_4) \equiv k_4 + 1$ (modulo 4967).

Las expresiones para simplificar con lenguaje C
modulo(33288857 + 1,4967), que al simplificar da $g(k_4)$ = 24; modulo(1526+24,4969), que al simplificar calcula $h_1(k_4)$ y se obtiene que $h_1(k_4)$ =1551.

Puesto que esta posición esta libre, se la puede asignar al cuarto archivo, el quinto y el sexto van a posiciones libres. Hay otras dobles asignaciones para k_7, entonces $h(k_7) \equiv 1551 \pmod{4969}$ y $h_1(k_7) \equiv h(k_7) + g(k_7) \equiv 1551+3265 = 4716 \pmod{4969}$, posición que ya estaba ocupada por $h(k_5)$, por lo tanto hay que continuar $h_2(k_7) \equiv h(k_7) + 2g(k_7) \equiv 3112 \pmod{4969}$, que esta libre. En este caso se obtiene la siguiente tabla:

CODÍGOS	H(K)	$H_1(K)$	$H_2(K)$
k_1= 70560069	269		
k_2 = 49731278	1526		
k_3= 29556190	578		
k_4 =33288857	1526	1551	
k_5= 10191166	4716		
k_6 = 33739186	4645		
k_7= 16349561	1551		3112
k_8 = 5119758	1688		
k_9 =13660359	578	4716	
k_{10} = 28637873	1526	1688	2798
k_{11} = 505576452	578	1002	

Tabla 1. Secuencia posibles de las direcciones

Con el siguiente código se calculan los datos de la tabla anterior.

```
#include<stdio.h>
#include<conio.h >
#include<math.h>
int n=11,i,j,c=0;
double cedula[11]={70560069,49731278,29556190,
3328857
,10191166,33739186,16349561,5119758,13660359,2863
7873,505576452};
```

```
void repmo(double[],int);
void moprim(double[],int);
void modulo(double[],int);
void modu2(double[],double [],double[],int );
void modulo3(double [],double []);
void modulo4(double[],double[]);

void main()
{clrscr();
 gotoxy(2,1); printf(" # CEDULA\n\n ");
 for(i=0;i<11;i++)
 {printf(" %0.0f\n ",cedula[i]);
  getch();
  }
 modulo(cedula,0);
 }
void modulo(double r[],int a)
{double copia[11];
 for(i=0;i<11;i++)
 {          copia[i]=fmod(r[i],4969);
  }
 gotoxy(15,1); printf("1MODULO\n");
 moprim(copia,a+20);
 repmo(copia,a+40);
}

void repmo(double x[],int a)
{ double y[11];double m2[11];
            for (i=0;i<n;i++)
              {for(j=i+1;j<n;j++)
                   { if (x[i]==x[j])
                       y[i]=i;
                       m2[i]=x[i];
                         }
             }
            gotoxy(30,1);printf("2MUDULO") ;
            modu2(y,cedula,m2,a+40);
}
void modu2(double y[],double cedula[],double r[], int a)
{int i;double m[11];
 for(i=0;i<n;i++)
  {m[i]=fmod(cedula[i]+1.4967,4967);
             }
 modulo3(r,m);
}
void modulo3(double r[],double m[])
{int i;double w[11];
  for(i=3;i<n-1;i=i+3)
  {w[i]=fmod(r[i]+m[i],4965);
            gotoxy(32,i+3);printf("%0.0f\n",w[i]);
            getch();
   }
  modulo4(w,m);
  }

void moprim(double x[],int y)
{for(i=0;i<n;i++)
  {gotoxy(y,i+3);
            printf("%0.0f\n",x[i]);
             getch();
             }
 }
void modulo4 (double w[],double m[])
{int i;double v[11];
 gotoxy(42,1);printf("3MODULO");
for(i=5;i<n;i=i+4)
 {v[i]=fmod(w[i]+m[i],4969);
  gotoxy(42,i+3);printf("%0.0f\n",v[i]);
  getch();
  }
 }
```

3. CONCLUSIONES

Como se puede observar en el ejemplo la función Modulo permite minimizar memoria al guardar códigos de ocho digito con códigos de cuatro dígitos.

Otro aspecto importante es el uso de teoría de número en problema computacionales.

Se presentan algunos inconvenientes en la función de fragmentación, como técnica para almacenamiento de datos:

El tamaño de la tabla es fijo; por lo tanto, hay que hacer una evaluación previa del tamaño máximo de la tabla, el cual no podrá cambiarse dinámicamente; de hecho, aunque se conozca el tamaño máximo a priori, conviene sobredimensionarla siempre (aproximadamente en un 10%), para tener un buen rendimiento.

Si se pretender borrar códigos, además de realizar búsquedas e insertar códigos, las tablas de fragmentar no son recomendables por que habría que proceder a una reestructuración muy grande.

4. BIBLIOGRAFÍA

[1] Avellanas, M; Hernández, G. et al: Practicas de Álgebra con Derive. Facultad de informática. Universidad Politécnica de Madrid. 1992

[2] A.Baker. Breve introducción a la Teoria Analítica de Números. Alianza Editorial (1986).

[3] Evans,B; Jonson, J.: Uses of Technology in the Mathematics Curriculum. Cipher Systems. 1990I. Niven, H. Zuckerman e H. Montgomery, *An Introduction to the Theory of Numbers* 5th ed., New York, John Wiley & Sons, 1991

[4] G. Hardy e. Wright, *An Introduction to the Theory of Numbers* 5th ed., Oxford, Clarendon Press, 1979