TRABAJO DE INVESTIGACION E IMPLEMENTACION DEL ALGORITMO DE EUCLIDES Y ALGORITMO EXTENDIDO DE EUCLIDES

- Angel Villagomez Iquira

(algoritmo de euclides extendido, algoritmo de mcd por fuerza bruta)

- Marco Antonio Guillen Davila

(algoritmo de euclides, algoritmo binario)

- Henry Andrew Medina Condo
- Rodrigo Victor Vilca Montesinos

RESUMEN:

En este trabajo de investigación se realizó el análisis de diferentes algoritmos cuyo propósito es calcular el máximo común divisor de 2 números, se analizó el tiempo que demoran en calcular el mcd de 2 números, cabe resaltar que los números analizados eran minimo 128 bits.

INTRODUCCIÓN:

El objetivo de esta investigación es calcular cuál es el mejor algoritmo con respecto al tiempo de ejecución y su complejidad.

CONTENIDO TEÓRICO:

-algoritmo de euclides
-algoritmo binario
-algoritmo de euclides extendido
-algoritmo de mcd por fuerza bruta

Algoritmo de Euclides Clásica:

DEFINICIÓN:

- El algoritmo de Euclides es un procedimiento que nos permite calcular el máximo común divisor de dos números que se realiza mediante la división del mayor número con el número menor, si la división es exacta el mcd es el número menor. Si la división no es exacta, entonces se toma el residuo y se divide tantas veces como se pueda hasta que este llegue a cero, el mcd será el último número con el que se le puede dividir.

PSEUDO- ALGORITMO:

1. Selecciona dos números “a”, “b” , se tiene que cumplir que “b”>”a”.
2. Hallamos el residuo (r) de “b/a” y le damos ese valor a “a” mientras que “b” tomará el valor anterior de “a”.
3. Hacemos la misma operación hasta que el valor de “a” sea igual a cero, si ese es el caso retornamos “b” que representará nuestro mcd.

SEGUIMIENTO NUMÉRICO:

a	b	r
133	1234	37
37	133	22
22	37	15
15	22	7
7	15	1
1	7	0
0	1	

MCD=1

IMPLEMENTACIÓN EN C++:

```
ZZ euclides(ZZ a, ZZ b) {
        if (a == 0)
                return b;
        return euclides(mod(b, a), a);
}
```

Complejidad:

Dentro del programa nos encontramos con la siguiente operación, la cual se realiza hasta que el residuo sea cero, pero podemos observar que se cumple algo con respecto a los residuos.

$$a0 = a1 * q0 + a2 \Rightarrow a2 < \frac{a0}{2}$$
$$a1 = a2 * q1 + a3 \Rightarrow a3 < \frac{a1}{2}$$
$$a2 = a3 * q2 + a4 \Rightarrow a4 < \frac{a2}{2} < \frac{a0}{2^2}$$
$$a3 = a4 * q3 + a5 \Rightarrow a5 < \frac{a3}{2} < \frac{a1}{2^2}$$

Con esto obtenemos 2 propiedades:

$$a_{2*i} < \frac{a_0}{2^i} \qquad \text{y} \qquad a_{2*i+1} < \frac{a_1}{2^i}$$

Debido a que a0 es el mayor de ambos el máximo grado de complejidad que podría tener sería cuando :

$$log(max(a, b))$$

Por tanto el tiempo de complejidad será:

$$O(log(max(a, b)))$$

Algoritmo Binario de MCD:

DEFINICIÓN:

- El algoritmo binario de mcd como su nombre nos indica es un algoritmo que nos permite hallar el mcd de 2 números reemplazando las divisiones por shift aritméticos, comparaciones y restas.

PSEUDO- ALGORITMO:

1. Seleccionamos 2 números a y b.
2. Si ambos números son 0, regresará como mcd=0. Si uno de los números es cero el mcd será el otro número pues todos los números dividen a cero.
3. Si a y b son pares, estos serán divididos por shift por 1, lo cual significa que en sistema binaria se le quitara un cero de la derecha, lo cual equivale a la división por 2, junto con esto tendremos un contador k el cual nos permitirá saber el número de veces que se requirió para dividir por 2 esos números a la vez.
4. Si “a” sigue siendo par entonces se le dividirá a la mitad hasta que sea impar.
5. Una vez “a” sea completamente impar haremos lo mismo con “b”. Ambos siendo impar haremos que estos se resten, para que esto siempre si a llegase a ser mayor

que a “b” estos tendrán que cambiar de valores y seguir restando hasta que “b” sea 0, esto significa que “a” será el mínimo valor divisible común de ambos números, aunque este aún no es mcd.

6. Para hallar el valor de del mcd “a” se multiplicará con k el cual representa la cantidad de veces que ambos números se dividieron por 2, de esta manera obtenemos el mcd.

SEGUIMIENTO NUMÉRICO:

operación	a	b	k
1	16	34	0
2	8	17	1
3	4	17	1
4	2	17	1
5	1	17	1
6	1	16	1
21	1	1	1
22	1	0	1

MCD=1 << k = 2

IMPLEMENTACIÓN EN C++:

```
ZZ binary(ZZ a, ZZ b) {
  if (a == 0)
        return b;
  if (b == 0)
        return a;

  int k;
  for (k = 0; ((a | b) & 1) == 0; ++k) {
        a >>= 1;
        b >>= 1;
  }

  while ((a & 1) == 0)
        a >>= 1;
```

```
  do {
        while ((b & 1) == 0)
        b >>= 1;

        if (a > b)
        swap(a, b);

        b = (b - a);
  } while (b != 0);
  a = a << k;
  return a;
}
```

Complejidad:

$\frac{N}{2^x}$ Donde x representa el número de pasos.

Para que el programa termine se tiene que cumplir:

$$2^x \geq N$$

$$log_2(2^x) \geq log_2(N)$$

$$x \geq log_2(N)$$

Por lo cual el tiempo de complejidad del programa nos quedaría:

$$O(\log(N))$$

ALGORITMO MCD POR FUERZA BRUTA:

DEFINICIÓN:

En este algoritmo todos los números que van desde desde “n” hacia abajo hasta encontrar uno que divida “m” y “n” exactamente

PSEUDO- ALGORITMO:

1- Selecciona dos números m, n , se tiene que cumplir que m<n
2- Se iguala ”i” al número m.
3- mientras que se cumpla que m modulo i es diferente a 0 o que n módulo i también es diferente a 0. se iguala ”i” a ”i” menos 1 y este ciclo se repite hasta encontrar un número que divida a ambos números iniciales (m y n), al encontrarlo se imprimirá el resultado

SEGUIMIENTO NUMÉRICO:

m	n	i
10	255	10
		9
		8
		7
		6
		5

MCD=5

IMPLEMENTACIÓN EN C++:

```
ZZ fuerza_bruta(ZZ m, ZZ n)
{
  ZZ i=m;
  while(divis(m,i)!=0||divis(n,i)!=0)
  {
    i=i-1;
    cout<<"mcd: "<<i<<endl;
  }
  return i;

}
```

Complejidad:

Debido a que la operación es lineal, el tiempo de complejidad será O(n).

ALGORITMO DE EUCLIDES EXTENDIDO:

DEFINICIÓN:

Este algoritmo es una modificación del algoritmo extendido de euclides que nos permite expresar el máximo común divisor como una combinación lineal al añadir 2 variables que vendrían a ser "x" y "y" los cuales se pueden usar para hallar la inversa de un número

PSEUDO- ALGORITMO:

En este algoritmo se seleccionan 2 números "a" y "b" los cuales se igualan a las variable r1 y r2 respectivamente , además se iguala una variable "s2" a 0 y otra variable "s1" a 1, lo

mismo con las variables “t1”=0 y “t2”=1,mientras que r2 sea mayor a 0 el entero “q” se iguala a la división de “r1”entre “r2”. La variable entera r se iguala a la resta de “r1” menos el producto de “q” con “r2”,además se iguala “r1” con “r2”,lo mismo pasa con “r2” y “r”,en el siguiente paso “s” es el resultado de la resta de “s1”menos el producto de “q” con ”s2” , a “s1” se le iguala con “s2” y a “s2” con “s”. Para terminar con el ciclo a “t” es el resultado de la resta de “t1”menos el producto de “q” con ”t2” , a “t1” se le iguala con “t2” y a “t2” con “t”.
se iguala “s” a “s, se comprueba que “s” sea menor a cero y si es asi “s” se lo iguala a “s” módulo de “b” por último se devuelve “r1”.
en bibliografia yo creo que solo use lo que la profes

SEGUIMIENTO NUMÉRICO:

r1	r2	s1	s2	t1	t2	s	t
672	38	1	0	0	1	-	-
38	26	0	1	1	-17	1	-17
26	12	1	-1	-17	18	-1	18
12	2	-1	3	18	-53	3	-53
2	0	3	-19	-53	336	-19	336

MCD=2

IMPLEMENTACIÓN EN C++ :

```
ZZ euclides_ext(ZZ a, ZZ b)
{
   ZZ r1,r2,s1,s2,t1,t2,s,t;

         r1 = a, r2 = b,
         s1 = 1, s2 = 0;
         t1 = 0, t2 = 1;

        while (r2 > 0)
        {
                ZZ q = r1 / r2;
                ZZ r = r1 - (q * r2);
                r1 = r2; r2 = r;

                s = s1 - (q * s2);
                s1 = s2; s2 = s;

                t = t1 - (q * t2);
                t1 = t2; t2 = t;
        };
        s = s1;
```

```
        if (s < 0)
                s = divis(s, b);
        return r1;
}
```

TIEMPOS:

bits / algoritmos	128	512	1024	2046
Euclidiano	1 ms	3.4 ms	3.3ms	3.3 ms
Euclidiano extendido	3 ms	3.5 ms	3.7 ms	3.9ms
Binario	2.9 ms	3.5 ms	3.7ms	3.7 ms
Fuerza bruta	muy largo	muy largo	muy largo	muy largo

Conclusión:

Como era de esperarse a medida que aumenta la cantidad de bits el tiempo por ejecución también aumenta, sin embargo el algoritmo de euclides al tener menos tareas que realizar siempre mantiene un tiempo por ejecución menor al resto. Por otro lado tenemos a un algoritmo de fuerza bruta el cual al tener una tarea bastante pesada la cual era verificar si cada número divide exactamente a ambos números que se habían dado sobrepasaba el tiempo de los demás algoritmos por mucho. Adicional a esto los tiempos de complejidad de los 3 restantes son prácticamente iguales pero al momento de la práctica el algoritmo de euclides demuestra ser mucho más y esto se debe a que en el código este termina siendo más eficiente que el resto.
En conclusión el algoritmo más eficiente probado por el grupo sería el algoritmo de euclides tanto en tiempo de complejidad como en práctica.

ESPECIFICACIONES TÉCNICAS:

```
                BIOS: F.11
          Procesador: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (12 CPUs), ~2.6GHz
             Memoria: 8192MB RAM
Archivo de paginación: 13756MB usados, 7588MB disponibles
   Versión de DirectX: DirectX 12
```