Escuela Profesional de Ciencia de la Computación

LaTeX

Comparación de GCD

Raul Edgar Quispe Totocayo

2017

Índice general

Índice de cuadros

Capítulo 1

Algoritmo de Euclides clasico

1.1. Definición

El algoritmo de Euclides es un método antiguo y eficaz para calcular el máximo común divisor (MCD). Fue originalmente descrito por Euclides en su obra Elementos.este algoritmo consiste en :
1.Si b=0 entonces maximoComunDivisor(a,b)=a y termina el algortimo
2.En otro caso maximoComunDivisor(a,b)=maximoComunDivisor(b,r) donde r es el resto al dividir a entre b.Para calcular maximoComunDivisor(b,r) se utilizan las mismas reglas

1.2. Algoritmo

Recordemos que $mod(a, b)$ denota el resto de la división de a por b. En este algoritmo, en cada paso $r = mod(rn+1, rn)$ donde $rn+1 = c$ es el dividendo actual y $rn = d$ es el divisor actual. Luego se actualiza $rn + 1 = d$ y $d = r$. El proceso continúa mientras d no se anule.
Datos: $a, b \in Z \ / b \neq 0$
Salida: $mcd(a, b)$

$$
\begin{aligned}
& c = |a|, d = |b|; \\
& while\ d \neq 0\ do \\
& \quad r = mod(c, d); \\
& \quad c = d; \\
& \quad d = r; \\
& return\ mcd(a, b) = |c|;
\end{aligned}
\tag{1.1}
$$

1.3. Seguimiento del codigo

a	b	q	r
957349573453465	8346583456	114699	4797633721
8346583456	4797633721	1	3548949735
4797633721	3548949735	1	1248683986
3548949735	1248683986	2	1051581763
1248683986	1051581763	1	197102223
1051581763	197102223	5	66070648
197102223	66070648	2	64960927
66070648	64960927	1	1109721
64960927	1109721	58	597109
1109721	597109	1	512612
597109	512612	1	84497
512612	84497	6	5630
84497	5630	15	47
5630	47	119	37
47	37	1	10
37	10	3	7
10	7	1	3
7	3	2	1
3	1	3	0

Cuadro 1.1: seguimiento de codigo

1.4. Implementacion del algoritmo

```
1 ZZ euclides(ZZ a, ZZ b)//
2 {
3     ZZ q,r;
4     q=a/b;
```

```
5       //r=module(a,b);
6       r=a%b;
7       while(r!=0)
8       {
9           q=a/b;
10          //r=module(a,b);
11          r=a%b;
12          a=b;
13          b=r;
14      }
15      return r;
16  }
```

Capítulo 2

Algoritmo de Menor Resto

2.1. Definición

es muy apropiada para fines teóricos. Que el resto sea positivo es adecuado, como vimos, para mostrar unicidad. Sin embargo el resto no tiene porque ser positivo, por ejemplo si a = 144 y b = 89,

- $144 = 89 \cdot 1 + 55$, resto r2 $= 55 < b = 89$
- $144 = 89 \cdot 2$ 34, resto r1 $= 34 < b = 89$

2.2. Implementacion del algoritmo

```
1  #include <iostream>
2  #include <NTL/ZZ.h>//includ  the lib ntl
3  #include <ctime>
4  #include <fstream>
5  using namespace std;//using std namespace
6  using namespace NTL;//using NTL namespaces
7  unsigned t0,t1;
8
9  ZZ module(ZZ &x,ZZ &y){
10     ZZ q=x/y,r;
```

```
11      if(q<0)
12      {
13          q=-1*q;
14          q++;
15          q=-1*q;
16          r=x-(q*y);
17      }
18      else
19      {
20          r=x-(q*y);
21      }
22      return r;
23  }
24  ZZ menor(ZZ x,ZZ y)
25  {
26      if(y<x)
27          return y;
28      else
29          return x;
30  }
31  void euclides(ZZ a, ZZ b)//
32  {
33      //t0=clock();
34      ZZ q,q1,r;
35      q=a/b;
36      q1=q;q1++;
37      r=menor(a-(q*b),a-(q1*b));
38      //r=a%b;
39      while(r!=0)
40      {
41          q=a/b;
42          q1=q;q++;
43          r=menor(a-(q*b),a-(q1*b));
44          //cout << a  << '\t'<<  " = " << q << "(" << b << ")"
               << "+" << r << endl;//print the euclides algorithm
```

```
45          a=b;
46          b=r;
47      }
48      cout << "result is "<< a << endl;
49      //t1=clock();
50  }
51  //funcion para guardar los datos
52  int main()
53  {
54      ZZ a,b;
55      cout << "input a:" ; cin >> a; //imput the numbers
56      cout << "input b:" ; cin >> b;
57      euclides(a,b);
58      //double time=(double(t1-t0)/CLOCKS_PER_SEC);
59      //cout << "Execution time:" << time << endl;
60
61  }
```

2.3. Seguimiento del codigo

a	b	q	r
9792347293422342	10048	974592347234	-356611584890
974592347234	-2	-356611584890	-95242407436
-356611584890	4	-95242407436	-70884362582
-95242407436	2	-70884362582	-24358044854
-70884362582	3	-24358044854	-22168272874
-24358044854	2	-22168272874	-2189771980
-22168272874	11	-2189771980	-270553074
-2189771980	9	-270553074	-25347388
-270553074	11	-25347388	-17079194
-25347388	2	-17079194	-8268194
-17079194	3	-8268194	-542806
-8268194	16	-542806	-126104
-542806	5	-126104	-38390
-126104	4	-38390	-10934
-38390	4	-10934	-5588
-10934	2	-5588	-5346
-5588	2	-5346	-242
-5346	23	-242	-22
-242	12	-22	0

Cuadro 2.1: seguimiento de codigo

Capítulo 3

Algoritmo Euclides Extendido

3.1. Definición

El algoritmo de Euclides extendido permite, además de encontrar un máximo común divisor de dos números enteros $a\,y\,b$, expresarlo como la mínima combinación lineal de esos números, es decir, encontrar números enteros $s\,y\,t$ tales que $mcd(a,b) = as + bt$. Esto se generaliza también hacia cualquier dominio euclideano.

3.2. El algoritmo de Euclides Extendido

3.2.1. Fundamentos

- Usar el algoritmo tradicional de Euclides. En cada paso, en lugar de a dividido entre b es q y de resto r se escribe la ecuación $a = bq + r$
- Se despeja el resto de cada ecuación.
- Se sustituye el resto de la última ecuación en la penúltima, y la penúltima en la antepenúltima y así sucesivamente hasta llegar a la primera ecuación, y en todo paso se expresa cada resto como combinación lineal.

3.3. Implementación

```
ZZ euclides(ZZ &a, ZZ &b)//
{
    ZZ q,r,r1=a,r2=b,s,s1,s2,t,t1,t2;
    s1=1,s2=0,t,t1=0,t2=1;
    while(r2>0)
    {
        q=r1/r2;
        r=r1-(q*r2);
        r1=r2;
        r2=r;
        //-----
        s=s1-(q*s2);
        s1=s2;
        s2=s;
        t=t1-(q*t2);
        t1=t2;
        t2=t;
    }
    return s1;
}
```

3.4. Seguimiento del algoritmo

q	r1	r2	r	s1	s2	s	t1	t2	t
9998111	93475	70209	70209	0	1	1	1	-9998111	-9998111
1	70209	23266	23266	1	-1	-1	-9998111	9998112	9998112
3	23266	411	411	-1	4	4	9998112	-39992447	-39992447
56	411	250	250	4	-225	-225	-39992447	2249575144	2249575144
1	250	161	161	-225	229	229	2249575144	-2289567591	-2289567591
1	161	89	89	229	-454	-454	-2289567591	4539142735	4539142735
1	89	72	72	-454	683	683	4539142735	-6828710326	-6828710326
1	72	17	17	683	-1137	-1137	-6828710326	11367853061	11367853061
4	17	4	4	-1137	5231	5231	11367853061	-52300122570	-52300122570
4	4	1	1	5231	-22061	-22061	-52300122570	220568343341	220568343341
4	1	0	0	-22061	93475	93475	220568343341	-934573495934	-934573495934

Capítulo 4

Algoritmo Binario de Euclides

4.1. Definición

El algorirmo binario de GCD , tambien conociddo como algortimo de stein es un algoritmo que calcula el mayor divisor común de dos enteros no negativos. El algoritmo de Stein utiliza operaciones aritméti- cas más simples que el algoritmo euclidiano conven- cional; Reemplaza la división por cambios aritméti- cos, comparaciones y sustracciones. Aunque el al- goritmo fue publicado por primera vez por el fisico y programador israelı Josef Stein en 1967, [2] puede haber sido conocido en la China del siglo I. Opera con los siguientes teoremas

4.2. el algoritmo de binario gcd

- Si a,b son pares, $mcd(a, b) = 2mcd(a/2, b/2)$.
- Si a es par y b impar o viceversa, $mcd(a, b) = mcd(a/2, b)$ *o* $mcd(a, b/2)$.
- Si a,b son impares , $mcd(a, b) = mcd(|a - b|/2, r)$, donde $r = mina, b$;

4.3. Implementación

```
1
```

```
2  ZZ binary_gcd( ZZ u,ZZ v)
3  {
4    int i;
5    if (u == 0) return v;
6    if (v == 0) return u;
7    cout << "a" << '\t' << "b" <<'\t' << "i" <<  endl;
8    for (i = 0; ((u | v) & 1) == 0; ++i) {
9           u >>= 1;
10          v >>= 1;
11         //cout << u << '\t' << v << '\t' << i <<  endl;
12   }
13
14   while ((u & 1) == 0)
15     u >>= 1;
16
17   do {
18        //cout << u << '\t' << v <<'\t' << i <<  endl;
19        while ((v & 1) == 0)
20            v >>= 1;
21        if (u > v) {
22           ZZ t = v; v = u; u = t;}
23        v = v - u;
24      } while (v != 0);
25   return u << i;
26 }
```

4.4. Seguimiento del algoritmo

a	b	t
843563845	34534	0
17267	843546578	0
17267	421756022	0
17267	210860744	0
17267	26340326	0
17267	13152896	0
17267	85490	0
17267	25478	0
12739	4528	0
283	12456	0
283	1274	0
283	354	0
177	106	0
53	124	0
31	22	0
11	20	0
5	6	0
3	2	0
1	2	0

Cuadro 4.1: seguimiento de codigo

Capítulo 5

left-shift binary algorithm

5.1. Definición

Este algoritmo debe su nombre al hecho de que se hace multiplicación por 2. En representación binaria el efecto de multiplicar por dos es un desplazamiento (en una posición), de la representación binaria original, hacia la izquierda.

5.2. Implementacion del algoritmo

```
#include <iostream>
#include <cmath>
#include <ctime>
#include <fstream>
#include <NTL/ZZ.h>
using namespace std;
using namespace NTL;
unsigned t0,t1;
void guardar(ZZ a,ZZ b,ZZ times,ZZ aux, ZZ s)
{
    ofstream archivo;
    archivo.open("lsbgcd.csv",ios::app);
```

```
13      archivo << a <<"&"<< b<<"&"<<times<<"&"<< aux<<"&"<<s << "
           \\\\\\hline"<< endl;
14      archivo.close();
15  }
16
17  ZZ menor(ZZ x,ZZ y)
18  {
19      if(y<x)
20          return y;
21      else
22          return x;
23  }
24  ZZ potencia(ZZ a,ZZ b)
25  {
26      ZZ result;
27      result=1;
28      for (int i = 0; i < b; i++)
29      {
30           result=a*result;
31      }
32      return result;
33  }
34  ZZ  lsbgcd(ZZ u, ZZ v)
35  {
36      ZZ a,b,t,aux,s,two;
37      two=2;
38      a=abs(u);
39      b=abs(v);
40      cout << "a" << '\t'<< '\t' << "b" << '\t'<< '\t' << "t" <<
           '\t'<< '\t' << "aux" << '\t'<< '\t' << "s" << endl;
41      while(b!=0)
42      {
43          s=0;
44          while(b*potencia(two,s)<=a)
45          {
```

```
46              s=s+1;
47          }
48          s=s-1;
49          t=menor(a-b*potencia(two,s),b*potencia(two,s+1)-a);
50          a=b;
51          b=t;
52          if(a<b)
53          {
54              aux=a;
55              a=b;
56              b=aux;
57          }
58          cout << a << '\t'<< '\t' << b << '\t'<< '\t' << t << '
               \t' << '\t'<< aux << '\t'<< '\t' << s <<endl;
59          guardar(a,b,t,aux,s);
60      }
61      return a;
62  }
63  //save data
64  int main()
65  {
66      ZZ a,b;
67      cout << "input a:" ; cin >> a; //imput the numbers
68      cout << "input b:" ; cin >> b;
69      lsbgcd(a,b);
70  }
```

5.3. Seguimiento del codigo

a	b	t	aux	s
51459615586	94237394	51459615586	94237394	13
3210069858	94237394	3210069858	94237394	9
194473250	94237394	194473250	94237394	5
94237394	5998462	5998462	94237394	1
5998462	1737998	1737998	94237394	3
1737998	953530	953530	94237394	1
953530	169062	169062	94237394	0
277282	169062	277282	169062	2
169062	60842	60842	169062	0
60842	47378	47378	169062	1
47378	13464	13464	169062	0
13464	6478	6478	169062	1
6478	508	508	169062	1
1650	508	1650	508	3
508	382	382	508	1
382	126	126	508	0
126	122	122	508	1
122	4	4	508	0
6	4	6	4	4
4	2	2	4	0
2	0	0	4	1

Cuadro 5.1: seguimiento de codigo

Capítulo 6

Algoritmo Lehmer GCD

6.1. Algoritmo

El algoritmo de MCD extendido de Lehmer D. H. Lehmer propuso en un algoritmo dirigido a optimi- zar el cálculo del máximo común divisor (MCD) de dos enteros positivos de múltiple precisión utilizan- do mayormente operaciones de precisión simple, lo que permite utilizar en la mayor parte de los casos operaciones internas del procesador. Este algoritmo puede extenderse de manera directa a un algoritmo de MCD extendido, aplicable para encontrar la in- versa modular de sólo un dıgito. Además, observó que el proceso subyacente en el algoritmo de Eucli- des es la aplicación de sucesivas transformaciones lineales:

1 While y $\geq$ b do the following:

1.1 Set x_, y_ to be the high-order digit of x, y, respectively (y_ could be 0)

1.2 $A \leftarrow 1$, $B \leftarrow 0$, $C \leftarrow 0$, $D \leftarrow 1$

1.3 If $(grupos\_x == grupos\_y)$ esto se agregó al algoritmo del presente libro

1.4 While $(y\_ + C)\ ! = 0$ and $(y\_ + D)! = 0$ do the following:

- $q \leftarrow (x\_ + A)/(y\_ + C)$, $q\_ \leftarrow (x\_ + B)/(y\_ + D)$
- if $q = q\_$ then go to step 1.5
- $t \leftarrow A - qC, A \leftarrow C, C \leftarrow t, t \leftarrow B - qD, B \leftarrow D, D \leftarrow t$
- $t \leftarrow x\_ - qy\_, x\_ \leftarrow y\_, y\_ \leftarrow t$

1.5 If $B = 0$, then $T \leftarrow xmody$, $x \leftarrow y, y \leftarrow T$

- otherwise, $T \leftarrow Ax + By, u \leftarrow Cx + Dy, x \leftarrow T, y \leftarrow u$

2 Compute $v = gcd(x, y)$ using Algorithm 2.104

3 Return(v)

6.2. Implementación

```
#include <iostream>
#include <limits.h>
#include <NTL/ZZ.h>
#include <fstream>
#define MAX_POTENCIAS 3 // para generar un arreglo de
   potencias de la base
#include <module.h>//this is my library Module
using namespace std;
using namespace NTL;
void lehmer_gcd(ZZ x, ZZ y);
ZZ enuclides( ZZ a,  ZZ b);
void guardar(int a,int b,int times,int resultado,int x,int y)
{
    ofstream archivo;
    //string line = "\\\\hline";
    archivo.open("lehmer.csv",ios::app);
    archivo << a <<"&"<< b<<"&"<<times <<"&" << resultado<<"&"
        << x <<"&" << y <<  "\\\\\\hline"<<endl;
    archivo.close();
}
int main()
{
    double segs;
    ZZ x = 565642334;
```

```
23      ZZ y = 24423424;
24
25      cout << "Aplicando Lehmer, gcd(" << x << ',' << y << ")\n"
           ;
26      lehmer_gcd(x,y);
27
28      int dj;
29      clock_t t_ini = clock();
30      dj = euclides(x,y);
31      clock_t t_fin = clock();
32      cout << dj << " en " << (double)(t_fin - t_ini)*1000.0 /
           CLOCKS_PER_SEC;
33      return 0;
34
35  }
36
37  ZZ *genera_array_base(ZZ base);
38  ZZ digitos_base(ZZ num, ZZ arr[], ZZ &grupos);
39
40  void lehmer_gcd(ZZ x, ZZ y)
41  {
42      clock_t t_ini = clock();
43
44      ZZ x_, y_, a, b, c, d;
45      ZZ q, q_, t, tt, u;
46      ZZ grupos_x;
47      ZZ grupos_y;
48      ZZ base = 1000; // determina un arreglo de potencias de la
            base.
49      ZZ length_bits = sizeof(int)*CHAR_BIT;
50      ZZ *arr_potencias = genera_array_base(base); // se crea un
            array con las potencias de la base.
51                                              // si es base 2 se
                                                   trata de una forma
                                                   especial (moviendo
```

```
                                                bits)
52      while(y >= base){
53
54          x_ = digitos_base(x, arr_potencias, grupos_x ); // x_
               tendr  los d gitos m s significativos que unidos
                ser n <= a la base.
55                                  // la funci n usa b squeda
                                       binaria en el array de
                                       potencias  de la base
56          y_ = digitos_base(y, arr_potencias, grupos_y);
57
58          a = 1; b = 0; c = 0; d = 1;
59          if(grupos_x == grupos_y){ //necesario pues en la
               siguiente vuelta hay que asegurar que x e y tengan
               la misma cantidad d cifras
60              while( ((y_+c) != 0) && ((y_+d) != 0) ){
61                  q = (x_+a) / (y_+c);
62                  q_ = (x_+b)/ (y_+d);
63                  if (q != q_){
64                      break;
65                  }
66                  t=a-q*c;
67                  a = c;
68                  c = t;
69                  t = b - q*d;
70                  b = d;
71                  d = t;
72                  t = x_ - q*y_;
73                  x_ = y_;
74                  y_ = t;
75                  guardar(x_,y_,a,b,c,d);
76              }
77
78          }
79          if (b == 0){
```

```
80              tt = x%y;
81              x = y;
82              y = tt;
83          }
84          else{
85              tt = a*x + b*y;
86              u = c*x + d*y;
87              x = tt;
88              y = u;
89          }
90      }
91
92      clock_t t_fin = clock();
93
94      cout << "Lehmer redujo a gcd(" << x << ',' << y << ") en "
           ;
95      double segs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
96      cout << segs * 1000.0 << " milisegundos. \n" << '\n';
97
98      cout << "euclide  gcd(" << x << ',' << y << ") = ";
99
100     t_ini = clock();
101     ZZ v = euclides(x,y);
102     t_fin = clock();
103     cout << v << '\n';
104     segs += (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
105     cout << "\nTiempo total:" << segs * 1000.0 << "
           milisegundos" << '\n';
106 }
107
108 ZZ *genera_array_base(ZZ base)
109 {
110     //repetir el primer y  ltimo  elemento para
111     //  retornar lo deseado en la busq. binaria
112     //  as  se agregan dos elementos m s
```

```
113     ZZ *arr = new ZZ[MAX_POTENCIAS+2];
114
115     arr[0] = base;
116     ZZ potencia = 1;
117     for(ZZ i = 1; i != MAX_POTENCIAS+1; i++){
118         potencia *= base;
119         arr[i] = potencia;
120     }
121     arr[MAX_POTENCIAS+1] = potencia;
122
123     return arr;
124 }
125
126 ZZ b_binaria(ZZ num, ZZ arr[], ZZ low, ZZ high); // devuelve
       el valor de la
127                                                     //
                                                           potencia
                                                            <= num
                                                            dentro
                                                            de arr
128 ZZ digitos_base(ZZ num, ZZ arr[], ZZ &grupos)
129 {
130     grupos = b_binaria(num, arr, 0, MAX_POTENCIAS+2 -1);
131     if (grupos == 0)
132         grupos = 1;
133     else
134         if (grupos == MAX_POTENCIAS+1 )
135             grupos = MAX_POTENCIAS;
136     return num / arr [grupos];
137 }
138 ZZ b_binaria(ZZ x, ZZ arr[], ZZ low, ZZ high)
139 {
140     ZZ medio;
141     if (high > low)
142         medio= (high-low)/2 + low;
```

```
143     else
144         medio = (low-high)/2 + low;
145     if(arr[medio] == x || (low > high))
146         return (low-1); //medio == low, asi retorna la
                potencia menor de la base
147     else{
148         if (arr[medio] < x)
149             return b_binaria(x,arr,medio+1, high);
150         else
151             return b_binaria(x,arr,low,medio-1);
152     }
153 }
154 ZZ euclides(ZZ a, ZZ b)//
155 {
156     ZZ q,r;
157     q=a/b;
158     r=module(a,b);
159     //r=a%b;
160     while(r!=0)
161     {
162         q=a/b;
163         r=module(a,b);
164         a=b;
165         b=r;
166 
167     }
168 
169     return b;
170 }
```

6.3. seguimiento del codigo

x	y	a	b	c	d
104	65	0	1	1	-8
65	39	1	-8	-1	9
39	26	-1	9	2	-17
25	14	0	1	1	-1
14	11	0	1	1	-1
11	3	1	-1	-1	2

Cuadro 6.1: seguimiento de codigo

Capítulo 7

Comparación de algoritmos

7.1. Tiempo de ejecución

7.1.1. Euclides Clasico

a	b	time
938492342349723492374234723	6823234234232423234	0.000337
548674568475645867	534653645634	0.0004
2323423324	242342	7.1e-05
23423	2342	0.000105
234	123	3.7e-05

Cuadro 7.1: Eficiencia de ejecución

7.1.2. Euclides Menor Resto

a	b	time
938492342349723492374234723	6823234234232423234	0.000382
548674568475645867	534653645634	0.000232
2323423324	242342	0.000133
23423	2342	0.000106
234	123	0.000121

Cuadro 7.2: Eficiencia de ejecución

7.1.3. Binario Euclides

a	b	time
938492342349723492374234723	6823234234232423234	0.000513
548674568475645867	534653645634	0.000198
2323423324	242342	0.000101
23423	2342	6.9e-05
234	123	5.2e-05

Cuadro 7.3: Eficiencia de ejecución

7.1.4. Euclides extendido

a	b	time
938492342349723492374234723	6823234234232423234	0.000599
548674568475645867	534653645634	0.000333
2323423324	242342	0.000126
23423	2342	8.2e-05
234	123	0.000169

Cuadro 7.4: Eficiencia de ejecución

7.2. Comparacion de Tiempo de ejecucion

i	a	b
1	938492342349723492374234723	68232342342324232 34
2	548674568475645867	534653645634
3	2323423324	242342
4	23423	2342
5	234	123

Cuadro 7.5: comparacion de tiempo

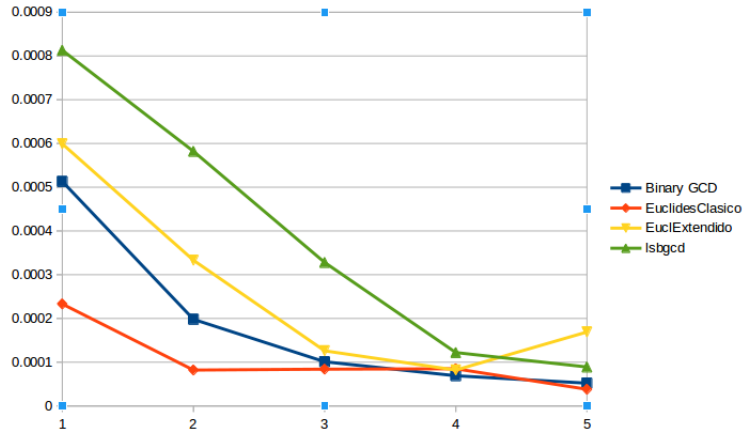

Figura 7.1: Caption conparacion de tiempo

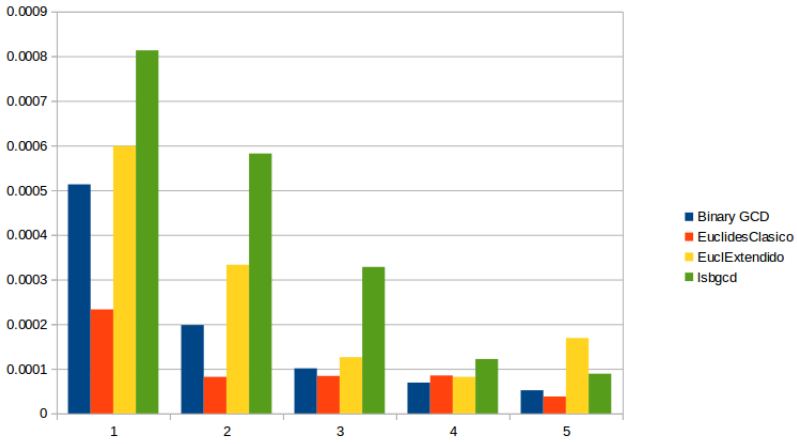

Figura 7.2: Caption comparacion de tiempo

Bibliografía

[1] Chapter 10. Number theory and Cryptography. URL http://ww3.algorithmdesign.net/sample/ch10-crypto.pdf.

[2] Cálculo del máximo común divisor:¿Porqué no se usa el algoritmo de Euclides?. Revista digital Matemática, Educación e Internet. Vol. 10, No 2. Marzo 2010. 22 de Agosto de 2013. URL http://www.tec-digital.itcr.ac.cr/revistamatematica/Secciones/Matematica_Algoritmos_Programacion/WMora_Calculo_del_maximo_comun_divisor_V10_N2_2010/WMora_Calculo_del_maximo_comun_divisor.pdf.

[3] Numerical Recipes in C : The Art of Scientific Computing. William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling.