Teoría avanzada de la computación

Entrega Final

Grado en Ingeniería Informática - 4.º Curso

Campus de Leganés Grupo 83

Carlos Gallego Jimenez
Carlos Rubio Olivares
Jorge Rodríguez Fraile

Índice

Introducción

En esta práctica introductoria se abordarán 2 problemas sencillos para ver su coste y complejidad computacional. El primero de ellos será el cálculo del máximo común divisor (MCD) y se solucionará usando dos algoritmos:

1. Descomponer ambos números en valores primos, y multiplicar los valores comunes de la factorización con su menor exponente.
2. Método de Euclides

El segundo problema consistirá en saber si un número es primo o no. En este caso el algoritmo será diseñado por nosotros.

Una vez hecho esto, el siguiente paso será el análisis y comparación de gráficas y resultados, ver qué algoritmos hacen un mejor trabajo, en qué situaciones y por qué.

Pruebas experimentales

En el primer ejercicio se han implementado dos algoritmos que son capaces de calcular el máximo común divisor de dos números. Para ello, se han utilizado la descomposición en factores primos y el método de Euclides.

Se han realizado 29 experimentos que se han repetido 10 veces cada uno para obtener estimaciones más robustas en cada uno de ellos se ha aumentado el valor de entrada exponencialmente por un valor de 2, obteniendo los siguientes resultados:

N	Algoritmo 1	Algoritmo 2	N	Algoritmo 1	Algoritmo 2
1	1122	1649	**18**	2,58E+15	2448
2	1026	1650	**19**	5,50E+15	4719
3	40692	2058	**20**	7,30E+15	2424
4	12508	4814	**21**	1,23E+17	4416
5	36327	2412	**22**	1,44E+17	82142
6	275176	2145	**23**	2,92E+17	3511
7	48563	3325	**24**	7,25E+17	8114
8	164975	2590	**25**	1,86E+19	10860
9	242158	5293	**26**	2,40E+19	2049
10	1487527	3688	**27**	7,33E+19	2954
11	576875	3898	**28**	1,28E+21	1709
12	810822	4436	**29**	2,28E+21	2201

13	2307418	3620	**30**	5,28E+21	2460
14	4485096	5886	**31**	1,15E+22	1815
15	8612786	4849	**32**	2,09E+23	1852
16	8291049	2561	**33**	3,79E+23	6861
17	8477569	6174			

Tabla 1. Resultados en nanosegundos del Ejercicio 1

En el segundo ejercicio se ha implementado un algoritmo que computa la primalidad de un número natural. En este caso, debido a que hay un gran número de pruebas, mostramos en la tabla únicamente las 64 primeras para evitar una tabla excesivamente grande, aunque se puede consultar en su totalidad en el Excel adjunto. En la gráfica se mostrarán estos primeros puntos.

N	Algoritmo 3	N	Algoritmo 3
0	4637	**3**	42976
0	1388	**3**	22134
0	1558	**3**	293107
0	1280	**3**	24203
0	1175	**3**	19120
0	1230	**3**	9943
0	1167	**3**	18447
0	1203	**4**	9887
0	1134	**5**	10344
0	1114	**4**	11730
1	87736	**5**	34902
1	7324	**5**	11766
1	6597	**5**	442690
2	19473	**4**	57092
2	8255	**5**	339093
1	7445	**5**	640190
2	7455	**5**	344070
2	11887	**5**	8630
2	7493	**5**	15628
2	7558	**6**	118305
2	51039	**5**	14535

2	18091	**5**	27976
2	18327	**5**	274044
3	209720	**5**	5455
3	12115	**6**	25174
3	27224	**5**	8588
3	11898	**5**	8582
2	25973	**6**	7587
3	25519	**7**	128811
2	10706	**6**	7101
4	446255	**7**	40932
3	11425	**6**	127749

Tabla 2. Resultados en nanosegundos del Ejercicio 2

Complejidad computacional

En este apartado se calculará la complejidad computacional de los algoritmos utilizados. Se recuerda que los dos primeros son referentes al problema 1 y el tercero, al problema 2.

- **Algoritmo 1 (utilizando descomposición en números primos):** En este algoritmo se utiliza un bucle for y un while anidado a este. De esta manera al hacer la factorización vemos si van coincidiendo los factores de ambos números. Si se da el caso, los multiplicamos por el mcd actual. Como se puede ver, la complejidad computacional de este algoritmo ha resultado ser $12x^2 + 12n + 4$.

  Véase: [1] Anexo. Algoritmo 1 para el máximo común divisor.

- **Algoritmo 2 (algoritmo de Euclides):** Este algoritmo necesita poca explicación, ya que es un algoritmo muy representativo y ya ha sido traducido al código infinidad de veces. Se puede ver que la complejidad es una orden menor que en el algoritmo 1 siendo de $11n + 7$

  Véase: [2] Anexo. Algoritmo 1 para el máximo común divisor.

- **Algoritmo 3 (algoritmo realizado por el equipo):** Este último algoritmo ha sido desarrollado por nosotros y simplemente nos dice si un número es primo o no. El funcionamiento es bastante simple, recorre todos los números del 2 hasta (n-1)/2, en el caso de que alguno de estos números sea divisible por el número de entrada, se devuelve False. La complejidad del algoritmo resulta ser de $9n + 7$

  Véase: [3] Anexo. Algoritmo 3 para el cálculo de números primos.

Estudio de la complejidad computacional de los algoritmos implementados

Después de realizar los experimentos se ha decidido comparar los resultados de forma gráfica que utiliza una escala logarítmica en base 10 para una mejor visualización:

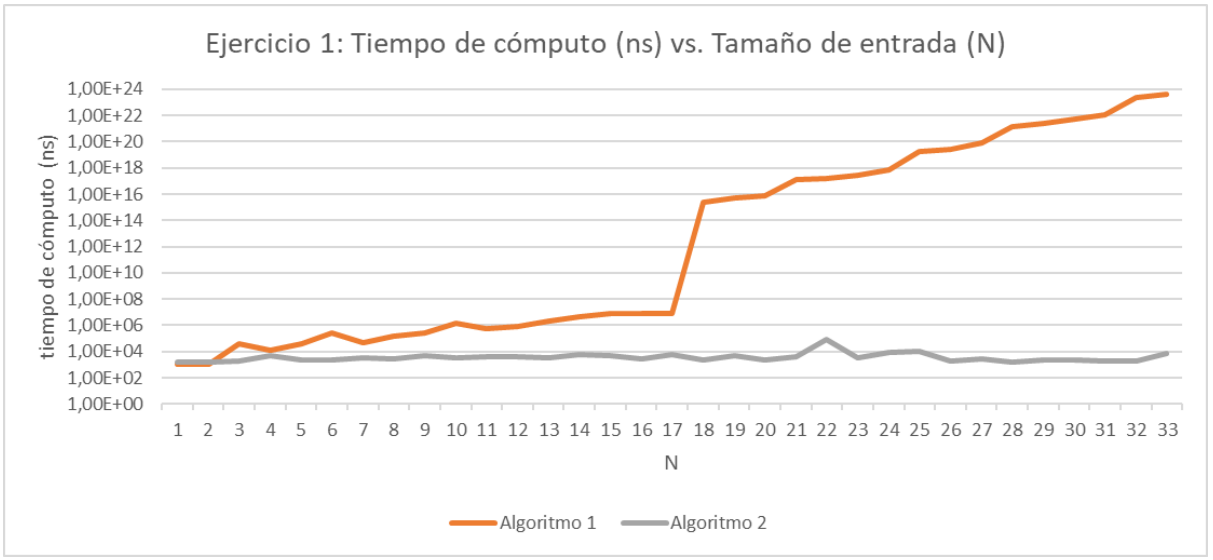

Ilustración 1. Gráfica de comparación de tiempos (nanosegundos) con respecto al tamaño de entrada N para el Ejercicio 1

Como se puede observar, el algoritmo 1 es considerablemente más costoso que el algoritmo 2. Esto se ha podido prever en el apartado anterior en el cálculo de complejidades. Como puntos importantes, se puede ver que hay un salto en tiempo de ejecución notable cuando se alcanza la mitad de la experimentación. Esto se puede deber a que se produce un cambio en el formato de los números, pasando de 32 a 64 bits. En cuanto al resto de la gráfica, la evolución del tiempo de ejecución de ambos algoritmos es suficientemente representativa para poder asumir que el algoritmo 2 es más eficiente.

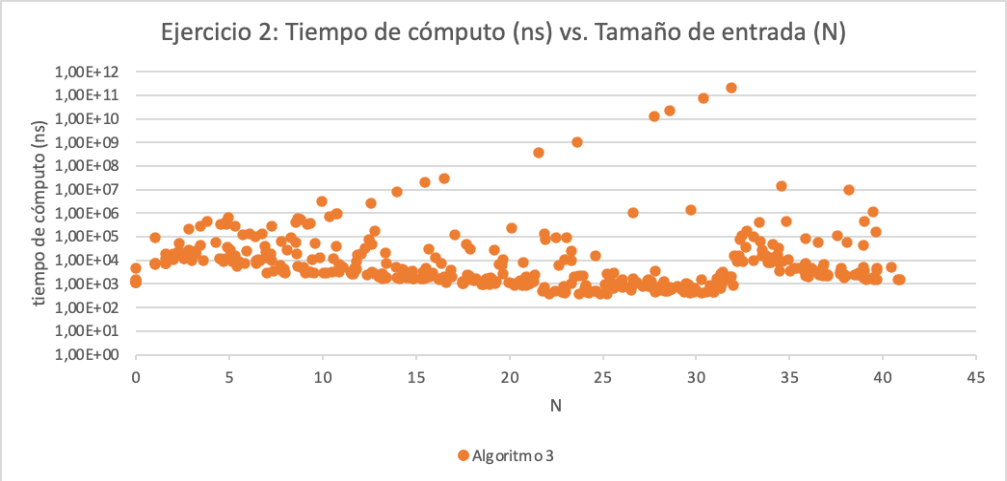

Ilustración 2. Gráfica comparativa entre tiempo de cómputo (nanosegundos) con respecto al tamaño de entrada N para el Ejercicio 2

En la segunda gráfica se puede ver como hay dos tendencias diferenciadas. En primer lugar, se puede observar el caso base, en el que el algoritmo es capaz de calcular con un tiempo de cómputo menor la primalidad de los números, esto sucede porque estos números o son primos pequeños o son no primos, al dividirse por otros primos encontramos rápidamente la solución.

En segundo lugar, se puede observar que según aumentamos la dimensión de la entrada hay una tendencia ascendente en tiempo de cómputo para ciertas instancias, esto se debe a que estas instancias son números primos y la complejidad de descubrir su primalidad aumenta, ya que el número de operaciones a realizar es mayor.

Complicaciones y dificultades

En cuanto a las complicaciones encontradas en el transcurso de la práctica no ha habido casos relevantes debido a que la implementación de estos algoritmos es medianamente sencilla y aceptable.

En caso de haber algo digno de mención, sería el hecho de la generación de resultados, debido a que con algunos algoritmos se tenían tiempos de ejecución altos por lo que, o bien hemos tenido que acotar el dominio del problema o ajustar los resultados.

Conclusiones generales

Para finalizar, está práctica inicial nos ha servido para conocer la dinámica de la asignatura, con qué clase de problemas trabajaremos y familiarizarnos con conceptos como la complejidad computacional. También hemos podido ver lo influyente que es desarrollar un código que juegue con estos términos y pueda aprovechar al máximo la velocidad de procesamiento de la máquina y tener resultados en el menor tiempo posible.

Anexo

[1] Algoritmo 1 para el máximo común divisor.

```
public BigInteger algoritmo1(BigInteger number1, BigInteger number2) {
    BigInteger number3 = number1.min(number2);
    BigInteger mcd = BigInteger.valueOf(1);
    number3.add(BigInteger.valueOf(1));
    for (BigInteger i = BigInteger.valueOf(2); i.compareTo(number3) == -1; i = i.add(BigInteger.valueOf(1))) {
        //T(ini) + T(cond) + n_iter*(T(cond) + T(incr) + T(blq)) = 2 + 2 + n*(2 + 3 + 12n+7) = 12n^2 + 12n + 4
        while (number1.mod(i).equals(BigInteger.valueOf(0)) && number2.mod(i).equals(BigInteger.valueOf(0))) {
            //T(cond) + n_iter*(T(bloq)+T(cond)) = 7 + n*(6+7) = 12n+7
            number1 = number1.divide(i); //2
            number2 = number2.divide(i); //2
            mcd = i.multiply(mcd); //2
        }
    }
    return (mcd);
}
```

Ilustración 3. Código y estudio de la complejidad del Algoritmo 1

[2] Algoritmo 2 para el máximo común divisor.

```
public BigInteger algoritmo2(BigInteger number1, BigInteger number2) {
    while (!number1.equals(BigInteger.valueOf(0)) && !number2.equals(BigInteger.valueOf(0))) {
        //T(cond) + n_iter * (T(bloq) + T(cond)) = 7 + n*(4+7) = 11n+7
        if (number1.compareTo(number2) == -1) { //T(cond) + Max(T(if),T(else)) = 2 + 2 = 4
            number2 = number2.mod(number1); //1+1 = 2
        } else {
            number1 = number1.mod(number2); //1+1 = 2
        }
    }
    return number1.max(number2);
}
```

Ilustración 4. Código y estudio de la complejidad del Algoritmo de Euclides

[3] Algoritmo 3 para cálculo de números primos.

```
public boolean isPrime(BigInteger n) {
    if (n.compareTo(BigInteger.valueOf(1)) < 1)
        return false; // 1

    for (BigInteger i = BigInteger.valueOf(2); i.compareTo(n.divide(BigInteger.valueOf(2))) < 1; i = i
            .add(BigInteger.valueOf(1))) { // n
        if (n.mod(i).compareTo(BigInteger.valueOf(0)) == 0)
            return false; // 1
    }

    return true;
}
```

Ilustración 5. Código y estudio de la complejidad del Algoritmo 3