Práctica de Complejidad Computacional

Primality Test - AKS

1. Introducción

El problema de determinar si un determinado número es primo ha interesado a los matemáticos desde la antigüedad, y hoy en día tiene múltiples aplicaciones en el ámbito de la Ciencia de la Computación.

Uno de los métodos más conocidos recibe el nombre de "*división por tentativa*", que consiste en ir dividiendo el número que se prueba por los sucesivos números primos menores que él, hasta alcanzar su raíz cuadrada. El método no es eficiente para números grandes, debido a su elevada complejidad computacional.

Hoy en día existen dos tipos de algoritmos para determinar si un número es primo: los llamados "*polinómico determinista"* y "*polinómico probabilístico"*. Los primeros nos dan la certeza absoluta sobre si un determinado número es primo. Los segundos están sujetos a un error muy pequeño (es decir, se determina que el número es un primo probable), pero tienen la ventaja de ser más rápidos. Entre estos últimos se encuentra el algoritmo de Miller-Rabin.

La utilización de números primos y la verificación eficiente de tal condición tiene consecuencias muy importantes en el campo de la criptografía. Sirva como ejemplo, el algoritmo de cifrado RSA, diseñado por los investigadores de MIT Rivest, Shamir y Adleman. Se trata de un sistema asimétrico que utiliza dos claves: la clave privada y la clave pública. El emisor de un mensaje, cifra este con su clave privada (publica); el receptor lo descifra con la clave pública (privada) del emisor. Tanto la clave pública como la privada son función de dos números primos p y q, muy grandes (mayores que 10^{100}). Para acceder al secreto de una comunicación cifrada con el algoritmo RSA, las técnicas de criptoanálisis intentan descubrir la clave privada a partir de la clave pública. Para ello buscará obtener p y q a partir de su producto f (componente de la clave pública), es decir, tratará de descomponer f en sus factores primos. Sin embargo, la elección de números primos muy grandes garantiza que es extremadamente difícil lograr la referida descomposición, lo cual constituye la fortaleza del sistema.

En 2002 M. Agrawal, N. Kayal y N. Saxena, del Instituto Tecnológico de Kanpur, en la India, publicaron un algoritmo determinista de complejidad computacional de orden polinómico que prueba si un determinado número es primo, el algoritmo denominado AKS [1]. El algoritmo está basado en el *pequeño teorema de Fermat.* Este en una forma muy resumida, enuncia lo siguiente:

> *si* ***p*** *es primo, y se cumple que* ***p*** *y* ***a*** *son primos entre sí, entonces* $\boldsymbol{a^p - a}$ *es divisible por* ***p****.*

si p es primo, y se cumple p y a son primos entre sí, entonces a^p-a es divisible por p.

Cabe destacar que el teorema, aunque puede aplicarse a números muy grandes, **establece una condición necesaria, pero no suficiente, para determinar si un número es primo**.

2. Objetivo de la práctica

La práctica consistirá en un análisis de la complejidad del algoritmo AKS, utilizando el código que se proporciona, disponible en Aula Global. El referido análisis se realizará mediante:

(a) un estudio empírico y

(b) un estudio analítico.

Opcionalemente puede utilizarse un lenguaje de programación diferente a Java, en cuyo caso los alumnos deberán codificar el algoritmo [1] para comparar los resultados con los obtenidos en Java.

Estudio Empírico

El objetivo del estudio empírico será **determinar experimentalmente *T(n)***. Debe considerarse el tamaño de entrada como el número de dígitos del número.

Será preciso llevar a cabo las siguientes actividades, que se reflejarán en los correspondientes apartados de la memoria:

Será preciso realizar pruebas de ejecución (desde valores pequeños hasta los valores máximos que permita la representación de números), midiendo el tiempo de ejecución. Se deberá:

1. Tabular los resultados y generar las gráficas pertinentes con los resultados de las pruebas de ejecución.
2. Determinar empíricamente la complejidad temporal *O(n)* del algoritmo a partir de las pruebas de ejecución.

Estudio Analítico

El objetivo del estudio analítico será determinar la complejidad temporal, ***T(n) y O(n),*** examinando los distintos pasos del algoritmo y deduciendo los componentes de coste computacional que contribuyen a *T(n)*.

Será preciso realizar las siguientes actividades:

1. Describir el enfoque utilizado para el análisis: premisas, fórmulas, teoremas, cuestiones básicas y sus respuestas, etc.
2. Realizar el análisis, **separando claramente los bloques de código analizados** para llegar a una expresión que nos indique la complejidad del algoritmo.

Observación

Para realizar los estudios se recomienda contestar a las preguntas que aparecen en el documento *CuestionesBásicasAKS*, disponible en Aula Global. La preguntas y respuestas pueden incluirse como parte de las deducciones que aparezcan en la memoria.

[1] Deberá ser posible contar con las funcionalidades que proporciona la clase *Big Integer* de Java: realizar cómputos con un numero de dígitos [mucho] mayor que los tipos estándar del lenguaje.

3. Entrega y evaluación

Se ha de entregar una memoria descriptiva que responda a las cuestiones planteadas, así como a otras cuestiones que surjan en el desarrollo de la misma (indicadas en apartados o subapartados adicionales). Además, si los alumnos lo consideran necesario, la memoria podrá contener apéndices relativos a secciones o resúmenes de código fuente.

Formato y Fecha de la entrega

Se plantearán una serie de Hitos que debéis abordar y resolver en las sucesivas semanas. La fecha de entrega final se indicará por separado en Aula Global. Dispondréis de un entregador semanal para acreditar el progreso de trabajo:

1. Hito 1: Análisis empírico de las heurísticas iniciales (pasos 1, 2 y 3).[2]

    a. Paso 1: Verificación de potencia perfecta.
    b. Paso 2: Cálculo de r.
    c. Paso 3: Cálculo del mcd.

2. Hito 2:

    a. Análisis empírico del paso 5, (*totient* y determinación de primalidad).
    b. Análisis empírico de la complejidad total.
    c. Estudio analítico de pasos 1, 2 y 3.

3. Hito 3: Estudio analítico del paso 5, *totient*, y de la determinación de la primalidad mediante el análisis de la condición suficiente (esto último sólo apoyándose en bibliografía).

    $Si\ (\ (X+a)^{num} \neq X^{num} + a\ (mod\ X^r - 1,\ num)\ )$, entonces COMPUESTO;

4. Entrega Final.

En las entregas debe incluirse el código empleado por los alumnos incluyendo las modificaciones realizadas. También deberá entregarse una memoria final en formato pdf. Se establece un límite máximo de páginas para la memoria con la siguiente distribución:

- Título: 1 página.
- Índice: 1 página.
- Introducción: ½ página.
- Estudio empírico: máximo 4 páginas.
- Estudio analítico: máximo 4 páginas.
- Conclusiones: ½ página.
- Referencias.

La Memoria irá en un fichero pdf con el nombre *Memoria_AKS_Grupo_Apellido1_Apellido2_Apellido3,pdf*

Entregad un fichero comprimido siguiendo la nomenclatura:
AKS_Grupo_Apellido1_Apellido2_Apellido3.zip, siendo *ApellidoK* el primer apellido, en orden alfabético ascendente, de cada uno de los integrantes del equipo, y *Grupo* el código asignado a cada grupo de trabajo.

[2] V. *AlgoritmoAKS*, en Aula Global.

Evaluación

La memoria deberá reflejar el trabajo realizado por los alumnos que componen el grupo.

Se evaluará el uso de tablas y figuras que refuercen las explicaciones y justificaciones de las respuestas.

También, si procede, se valorará la inclusión de referencias bibliográficas si se usan fuentes externas.

Se evaluará el cumplimiento de todas las indicaciones realizadas en todos los apartados de este enunciado.

Se permite la colaboración entre los alumnos, pero cada práctica y su correspondiente memoria deben ser desarrolladas y escritas de forma individual por el grupo. Al entregar la memoria los alumnos asumen que entienden y certifican que cumplen este requisito.

4. Referencias

[1] M. Agrawal, N. Kayal y N. Saxena, "PRIMES is in P", *Annals of Mathematics,* vol. 160, pp. 781-793, 2004.