Teoría avanzada de la computación

AKS - Entrega final

Grado en Ingeniería Informática - 4.º Curso

Campus de Leganés Grupo 83

Carlos Gallego Jimenez
Carlos Rubio Olivares
Jorge Rodríguez Fraile

Índice

1. Introducción

En este documento se detalla el análisis de la complejidad del algoritmo AKS de cálculo de la primalidad, realizando un análisis empírico y analítico de este. Para llevarlos a cabo se utilizará la documentación proporcionada y los conocimientos adquiridos, apoyando nuestros resultados con gráficos ilustrativos.

Para llevar a cabo un análisis más detallado del algoritmo, en la primera parte del documento se estudia la complejidad de las heurísticas que emplea para podar rápidamente aquellos números más sencillos y en la segunda parte, se lleva a cabo el análisis del núcleo del algoritmo.

2. Estudio Empírico

En esta parte del documento se hace un estudio empírico del algoritmo y las heurísticas que lo forman efectuando una serie de ejecuciones que nos permitan ver cómo se comportan respecto al tiempo con diferentes tamaños de entrada.

2.1. Heurísticas

Para analizar la complejidad del algoritmo, es necesario estudiar la complejidad de las heurísticas que se emplean para determinar la primalidad de un número. Las heurísticas se presentan en los siguientes apartados.

2.1.1. Verificación de potencia perfecta

La primera heurística que se usa en el algoritmo AKS es la verificación de la potencia perfecta. Esta heurística es utilizada para descartar que el número sea una potencia de un número entero.

Para llevar a cabo este análisis se han realizado experimentos con números primos de hasta 19 cifras, generando 10 números para cada cantidad de bits desde 1 hasta 64. Se han graficado según el valor del número N generado como se puede observar a continuación:

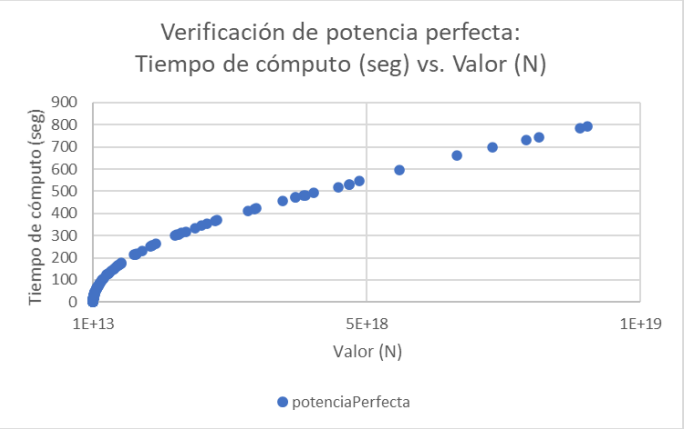


Figura 1. Gráfica comparativa entre el valor de entrada N y el tiempo de cómputo en segundos de la heurística verificación de la potencia perfecta del algoritmo AKS

Podemos ver que la gráfica es más precisa en los números más bajos dado que según aumenta el número de bits el rango de valores se va doblando, haciendo que esté menos poblada a la derecha en esta representación. Muestra una tendencia de raíz cuadrada ($\sqrt{n}$), por su forma más amplia que una logarítmica.

Para llevar a cabo un análisis más detallado también se ha decidido comparar el tiempo de cómputo con el tamaño de entrada, para ver una gráfica más descriptiva se ha decidido tener en cuenta la parte decimal del número de cifras según la cercanía a tener una cifra más. También se ha acotado el número de cifras de 15 a 19 dado que se pretende estudiar cómo se comporta la heurística con números grandes. Los resultados se pueden ver a continuación:

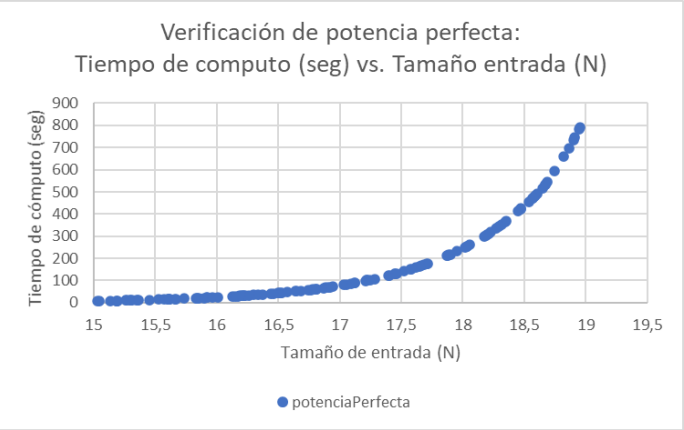


Figura 2. Gráfica comparativa entre el tamaño de entrada N y el tiempo de cómputo en segundos de la heurística verificación de la potencia perfecta del algoritmo AKS

Como se puede observar en la gráfica el tiempo de cómputo aumenta de manera exponencial según aumentamos el número de cifras, esto tiene sentido debido a que el eje horizontal de la gráfica pasa a ser el n.º de dígitos de la entrada, lo que genera una función exponencial.

2.1.2. Cálculo de r

En este apartado se procede al análisis empírico de la segunda heurística del algoritmo AKS que consiste en el cálculo de r que se utiliza para encontrar factores pequeños de n si existen.

Para realizar este análisis empírico se han realizado experimentos con números primos de hasta 19 cifras. En la primera gráfica se puede observar el tiempo de cómputo del cálculo de r frente al valor de la entrada, N.

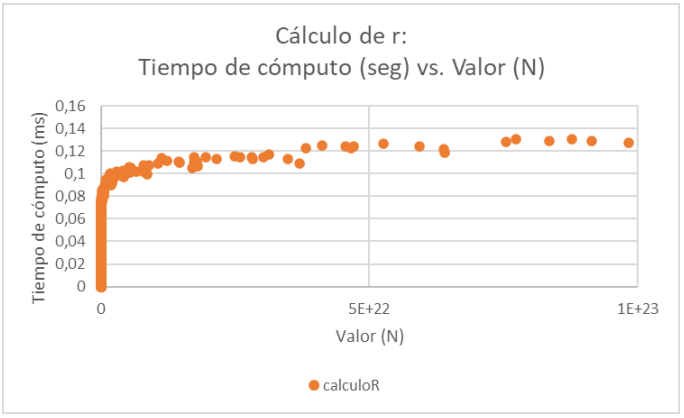


Figura 3. Gráfica comparativa entre el valor de entrada N y el tiempo de cómputo en milisegundos de la heurística cálculo de r del algoritmo AKS.

Por otra parte, centrándose en la tendencia que siguen los puntos puede verse de manera clara una función logarítmica que se estabiliza alrededor de 0.12 ms para este fragmento de los resultados.

Para realizar un análisis más detallado y siguiendo la estructura de análisis de la previa heurística se ha realizado otra gráfica en la que se compara el tiempo de cómputo con el número de cifras de la entrada.

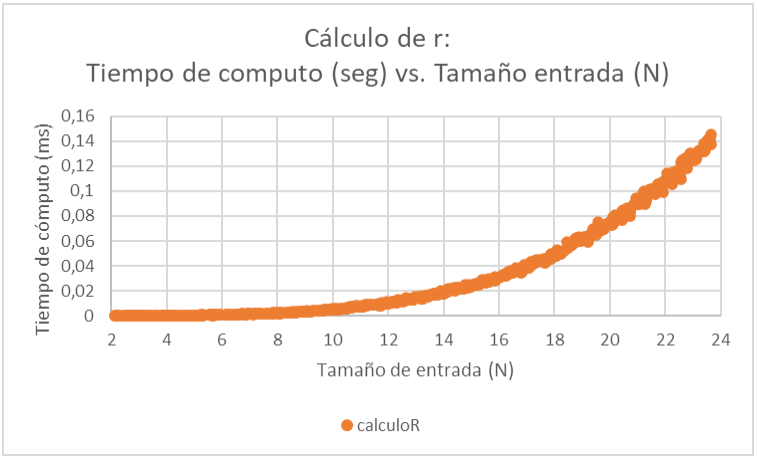


Figura 4. Gráfica comparativa entre el tamaño de entrada N y el tiempo de cómputo en milisegundos de la heurística cálculo de r del algoritmo AKS.

En esta última gráfica se observa una clara tendencia exponencial, cosa que era de esperar teniendo en cuenta resultados anteriores. En este caso se ha conseguido ver la tendencia de manera más clara, ya que ha sido posible mostrar más valores del eje x. Por otro lado, como siempre, debido a factores externos como el hardware, la clase BigInteger o el IDE a la hora de ejecutar se pueden observar ciertos outliers sobre todo al principio de la gráfica, después, los resultados empiezan a seguir la tendencia de manera más clara.

2.1.3. Cálculo del mcd

En estas 2 gráficas que se muestran a continuación se procede a analizar el cálculo del MCD. Se tiene en cuenta el valor y el tamaño de entrada.

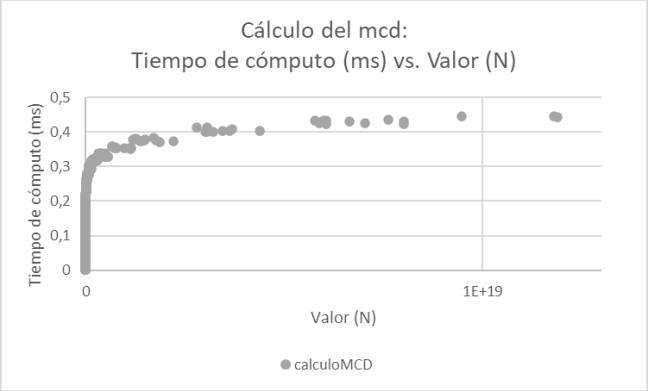


Figura 5. Gráfica comparativa entre el valor de entrada N y el tiempo de cómputo en milisegundos de la heurística verificación de MCD del algoritmo AKS

De la primera gráfica podemos ver que se ve un aumento de tiempo en los cálculos hasta valores cercanos a 5E10 donde se empieza a estabilizar. Se puede observar una tendencia logarítmica en la gráfica creciendo lentamente según aumentan los valores.

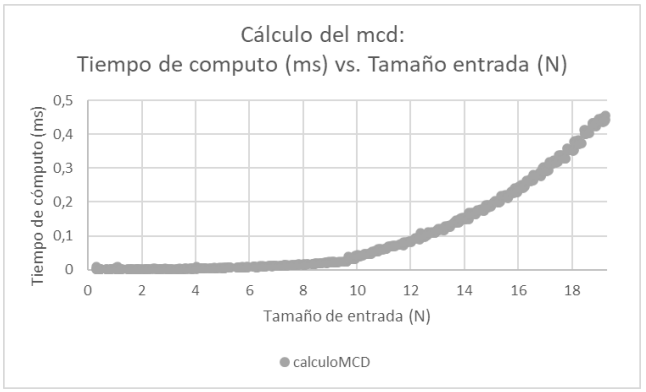
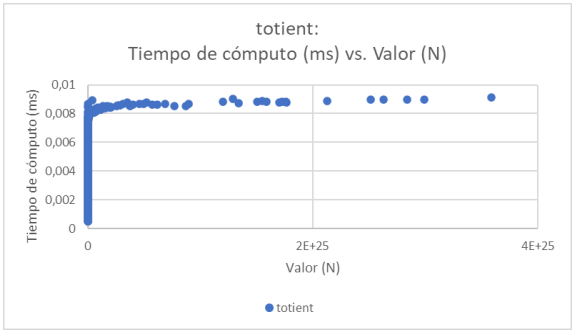


Figura 6. Gráfica comparativa entre el tamaño de entrada N y el tiempo de cómputo en milisegundos de la heurística verificación del MCD del algoritmo AKS

Por otro lado, en la segunda gráfica vemos una tendencia totalmente exponencial. Esto tiene sentido debido a lo explicado anteriormente en la verificación de potencia perfecta, cuando en el eje x se incluye el número de dígitos en lugar del valor numérico, la tendencia de la gráfica debería cambiar.

2.1.4. Totient y cálculo de primalidad

Este es el quinto paso del algoritmo AKS. Como en los apartados anteriores, se han usado dos gráficas, presentándose el tiempo de cómputo frente al tamaño de entrada y el valor.



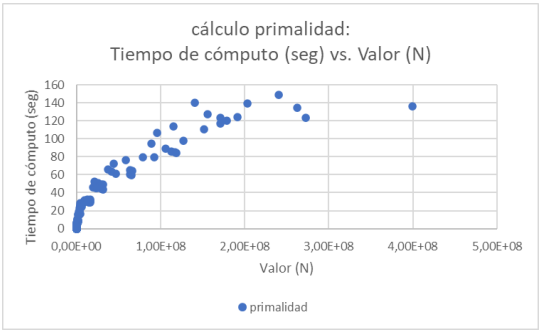


Figura 7. Gráfica comparativa entre el valor N y el tiempo de cómputo de los métodos de totient (en milisegundos) y cálculo de primalidad (en segundos) del algoritmo AKS

Estas gráficas muestran una clara tendencia logarítmica. En la gráfica del cálculo de primalidad, podemos ver unos tiempos mucho más altos, llegando algunos valores a los 150 segundos, también se puede observar una presencia mayor de ruido. A continuación se muestra el tiempo de computación frente al tamaño de entrada de estos mismos procesos.

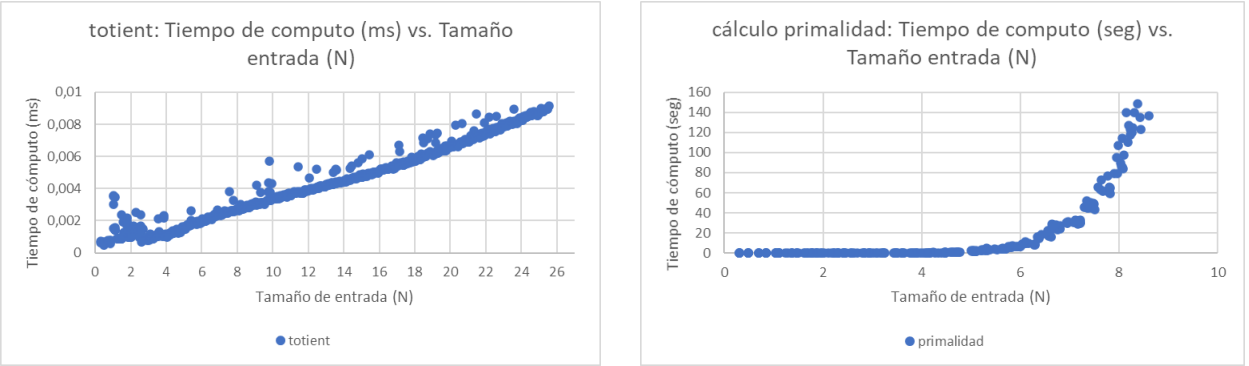


Figura 8. Gráfica comparativa entre el tamaño de entrada N y el tiempo de cómputo de los métodos de totient (en milisegundos) y cálculo de primalidad (en segundos) del algoritmo AKS

Por último se puede observar el tiempo de cómputo frente al tamaño de la entrada. La tendencia en este caso, como se lleva viendo en casos anteriores al cambiar la representación del eje x, la tendencia representada en la gráfica cambia. Podemos ver una presencia de ruido algo mayor en totient mientras que el cálculo de primalidad empieza a tener una dispersión mayor en etapas finales de la gráfica.

Como se ha mencionado anteriormente, estás gráficas son muy similares al apartado anterior, por lo que las mismas conclusiones se pueden sacar de aquí.

2.2. Algoritmo completo

En este apartado se realiza el análisis empírico de la complejidad total del algoritmo AKS. Para este fin, se utilizan dos figuras que representan el tiempo de cómputo frente al tamaño de la entrada y el número de dígitos de N que se representa de manera decimal para que la gráfica sea más descriptiva.

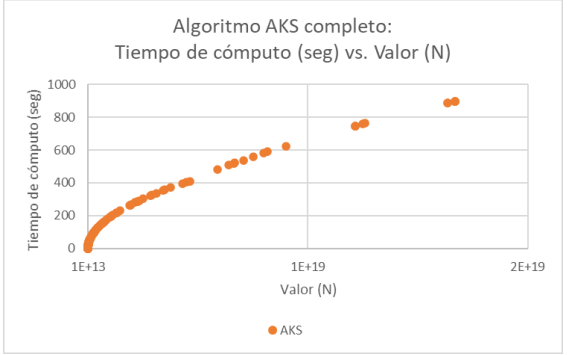


Figura 9. Gráfica comparativa entre el valor N y el tiempo de cómputo en segundos del test de primalidad AKS completo

Podemos comprobar que nuestros experimentos muestran una función logarítmica, pero no una base, sino una polilogarítmica ($log^k n$), por la velocidad en la que crece, que es superior a las otras vistas. En esta se puede ver claramente una tendencia sin ruido.

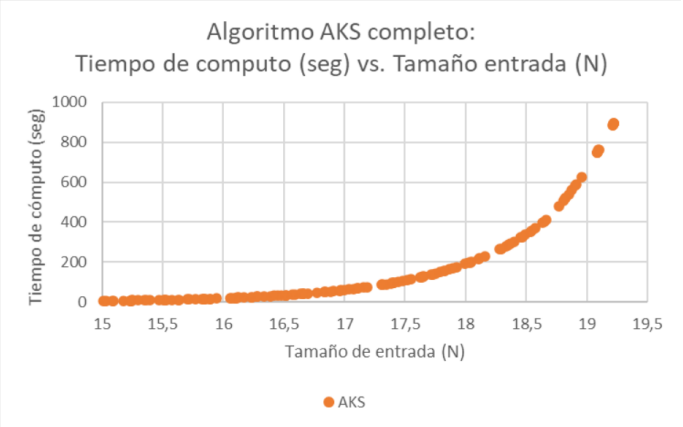


Figura 10. Gráfica comparativa entre el tamaño de entrada N y el tiempo de cómputo en segundos del test de primalidad AKS completo

En nuestra segunda gráfica podemos ver como el tiempo de computación escala exponencialmente cuando el número de dígitos en la entrada supera los 18 dígitos, ya que el número de operaciones que el algoritmo debe ejecutar aumenta proporcionalmente con la entrada al tener que verificar $\lfloor(\sqrt{\varphi(r)}log\ n\rfloor$ ecuaciones. Esta idea estará respaldada por su consecuente análisis analítico.

3. Estudio Analítico

En este segundo estudio se procede al análisis del código proporcionado, apoyándose en la documentación y resultados previos.

3.1. Potencia Perfecta

Se comienza el estudio analítico con el cálculo de la potencia perfecta. En este caso, se sabe según el lema 3.1 de la bibliografía [1] que para el cálculo de la potencia perfecta solo es necesario recorrer $\sqrt{n}$ números para conocer si n es una potencia perfecta, dado que las bases al cuadrado con las que se puede hacer no deben superar el valor a estudiar. Se usa la base al cuadrado, puesto que es el menor exponente que generaría un número compuesto. La prueba de exponentes se hace un máximo de veces, constante.

Este planteamiento concuerda con los valores representados en el estudio empírico. Se puede afirmar, por tanto, que las pruebas efectuadas han sido correctas, ya que están respaldadas por esta idea. Por lo que la complejidad es $O(cte\ *\sqrt{n}) = O(\sqrt{n})$.

Por otro lado, el uso de las cuestiones básicas ha sido de ayuda para concluir este apartado, por el hecho de que se ha podido comprobar el número de iteraciones que se hacen en el bucle del código de AKS.

3.2. Cálculo de r

Para calcular la complejidad de r, se debe analizar el código correspondiente a esta heurística. En el código podemos ver como las asignaciones y comparaciones se resuelven en tiempo constante, mientras que la parte más importante reside en el cálculo del *multiplicativeOrder* que por la bibliografía [1] (empleando la optimización de r) se sabe que se tendrán que probar $log^2(n)$ valores diferentes de r y esta complejidad corresponde a la complejidad del bucle. Para terminar dado que el cálculo de r lo forman dos bucles. Tenemos que para el bucle externo hay $log^2 n$ iteraciones, mientras que para el interno hay $log^2 n$, al combinar ambos bucles se obtiene $O(log^2 n * log^2 n) = O(log^4 n)$.

Al comparar estos valores con el análisis empírico se observa una gran similitud, ya que en la gráfica se sigue una tendencia logarítmica. Se puede asumir, por tanto, que los experimentos realizados son correctos.

3.3. MCD

Para la complejidad del MCD, se tiene un bucle for cuya complejidad será de $log\, n$ veces, que viene a raíz de que el peor caso es buscar el máximo común divisor de dos números consecutivos de la sucesión de Fibonacci.

Con esto se llega a que la complejidad vendrá dada por el número de iteraciones que se realicen en el bucle multiplicado por el coste $log\, n$ del MCD. Se obtiene $O(log^3 n)$ debido a que se prueban el peor caso $log^2 n$ valores de r para poder llegar al final del algoritmo.

3.4. Totient

En esta sección estudiaremos analíticamente el cálculo del totient dentro del algoritmo de AKS apoyándonos en la bibliografía. Para el estudio del totient lo primero que hay que tener en cuenta es que $\Phi(n)$ es la cantidad de números naturales que son menores o iguales que n y que son coprimos con n.

Para calcular el totient entonces se usa $\Phi(n) = n \cdot \prod_{p|n}(1 - \frac{1}{p})$, siendo *p* los distintos primos que dividen a n. En este cálculo en el peor caso se recorren todos los valores desde 2 hasta $\sqrt{n}$ en busca de divisores ($n > i^2$) con el for externo, y para el último divisor se realizan $log\, n$ divisiones ($i^k \leq n$) en el bucle. Por lo que el proceso entero tendrá una complejidad total en totient de $O(\sqrt{n}\, log\, n)$.

3.5. Cálculo de primalidad

Para el estudio analítico del cálculo de primalidad, utilizamos el lema que estipula que un número x es primo si y sólo si: $(x + a)^n \equiv (x^n + a)\ (mod\ n)$

A partir de este lema se obtiene una mejora: $(x + a)^n \equiv (x^n + a)(mod\ x^r + 1)$ donde está r es obtenida del paso 2. Esta mejora es factible, ya que el resto de $(x + a)^n : (mod\ x^r + 1)$ es equivalente al resto de $(x^n + a) : (mod\ x^r + 1)$.
Una vez estipulada la mejora de este lema, se toma el mejor caso de r tal que $O_r(n) > log^2 n$. Para examinar si un número es primo se necesita comprobar el valor de totient ecuaciones, cada una se estima que cuesta $O(r\ log^2 n) = O(log^4 n)$.
De nuevo, sobre este caso es posible utilizar una mejora, utilizando $\sqrt{\phi(r)}$ en lugar de $\sqrt{r}$, aunque en este estudio, esta mejora queda en algo anecdótico debido a que sobre todo se tiene en cuenta la cota asintótica superior, que acaba siendo logarítmica de igual manera. Este paso unido con el valor obtenido de totient hacen que el paso y la complejidad final de AKS sea $O(log^2 n * log^4 n) = O(log^6 n)$, dado que es la más alta entre la suma del resto de complejidades de los pasos.
Esta explicación analítica concuerda con los datos obtenidos en el estudio empírico en el apartado 2, ya que se presenta una clara tendencia logarítmica.

Todos estos datos, tanto teóricos como experimentales, indican que el estudio de este paso del algoritmo AKS ha sido correcto.

4. Conclusión

Este trabajo nos ha permitido ver cómo se realiza el estudio de un algoritmo para determinar cómo de costoso resultará computacionalmente, además, este test de primalidad cuenta con una estructura particular, dividida en heurística, que ha resultado especialmente útil para su estudio. Aunque AKS no sea el algoritmo más rápido actualmente para saber si un número es primo, es muy seguro y computacionalmente asequible, no como otros que se basan en probabilidad.

Además, los resultados que se han obtenido de manera analítica han concordado con los empíricos, por lo que se ha podido corroborar que los estudios iban bien encaminados y con una buena base práctica en la que sustentarlos.

5. Referencias

[1] M. Agrawal, N. Kayal and N. Saxena, "PRIMES is in P", *Annals of Mathematics*, vol. 160, no. 2, pp. 781-793, 2004. Available: 10.4007/annals.2004.160.781.

[2] R.Singer-Heinze ”Run Time Efficiency and the AKS Primality Test” Available: http://ccs.math.ucsb.edu/senior-thesis/Roeland-Singer.pdf