Programación de Dispositivos Móviles (2017-2018)
Grado en Ingeniería Informática
Universidad de Granada

Tutorial 1

Juan Alberto Martínez López
Alberto Armijo Ruiz

6 de abril de 2018

Índice

Introducción

Para esta práctica hemos desarrollado el algoritmo Miller-Rabin para decidir si un número es posible primo o no es primo, para ello hemos desarrollado dos versiones del algoritmo: Una en la que se realizan n aplicaciones para comprobar la primalidad y otro con una lista de números naturales con los que se comprueba el test para el primo y cada una de las bases de la lista. Para calcular la primalidad utilizamos el algoritmo de logaritmo discreto en el que comprobamos la existencia dado a,b y p de $\log_a(b) \mod n$ y si se cumple el número es primo.

1. Evaluación de tiempos

Para el algoritmo Miller-Rabin con 2000 aplicaciones podemos apreciar muy poco aumento del tiempo (Tabla 1.1), por lo que podemos concluir que el procesamiento de la primalidad no tiene una carga grande y no hay problema a la hora de calcular números grandes.

Número Primo	Tiempo ejecución (seg)
57347	0.008280
468577	0.009140
5555567	0.013386
87654337	0.012030
987654323	0.014271
3141592661	0.019104
11111111113	0.020818
121212121223	0.021233
2718281828489	0.025449
16180339892149	0.027566
800000000000017	0.031171

Tabla 1.1: Tabla tiempos de ejecución para el algoritmo Miller Rabin

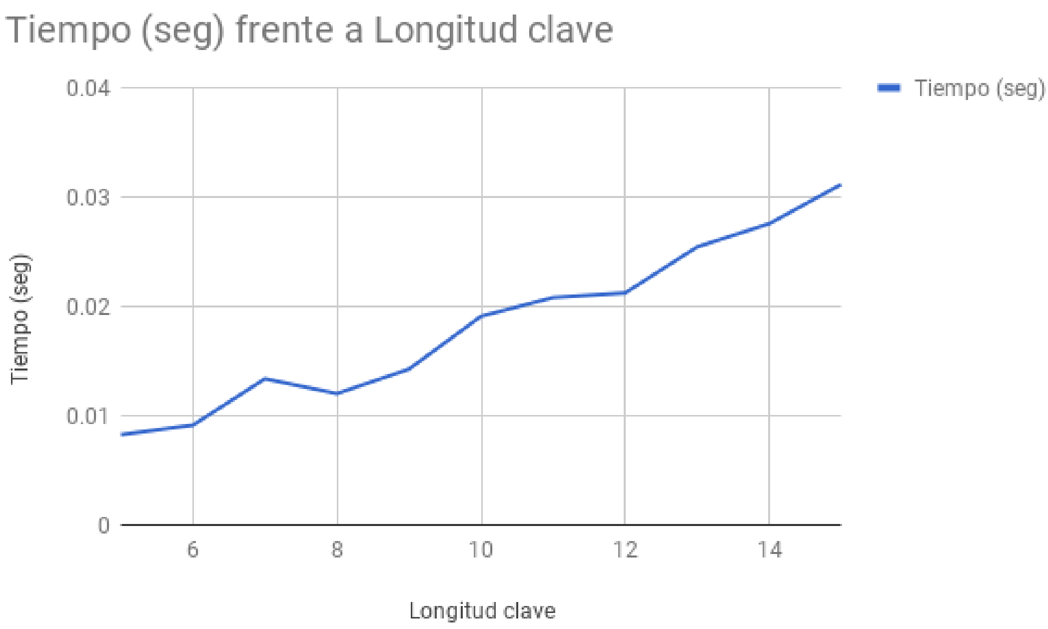

Figura 1.1: Tiempos vs Longitud clave Miller-Rabin

Para el algoritmo del logaritmo discreto se han calculado una a y c aleatoria y hemos calculado la b a partir de $b = a^c \mod p$. En las tablas 1.2 y 1.3 si que se puede apreciar un aumento considerable de los tiempos ya que de un tiempo a otro podemos ver como se cuadruplican de un resultado a otro.

En la siguiente figura podemos apreciar como el crecimiento del tiempo de ejecución tiene forma de una función exponencial.

Longitud clave	Tiempo ejecución (seg)
5	0.001187
6	0.004052
7	0.018117
8	0.080128
9	0.289543
10	0.73591
11	1.506318
12	5.264323
13	29.752512
14	76.744032
15	612.546521

Tabla 1.2: Tabla tiempos

A	B	P	Solución	Tiempos (s)
6	50628	57347	7	0.001187
8	449605	468577	11	0.004052
207	4374842	5555567	104	0.018117
4007	8515459	87654337	430	0.080128
40756	118205788	987654323	10748	0.289543
20544	253647140	3141592661	113	0.735911
112354	9048018943	11111111113	5658	1.506318
1245628	49579028347	121212121223	568985	5.264323
87569	1342094524016	2718281828489	5749833	29.752512
568236	14717101287551	16180339892149	389567512	76.744032
4555786	778596955901441	800000000000017	785951	612.546521

Tabla 1.3: Tabla análisis de logaritmo.

Figura 1.2: Tiempos vs Longitud clave

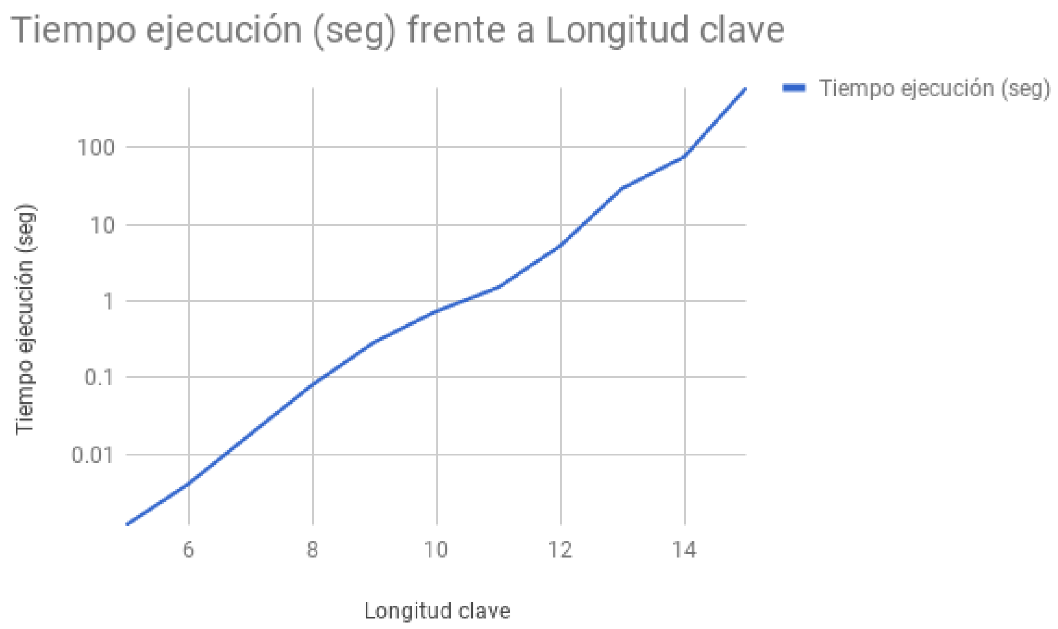

Figura 1.3: Tiempos vs Longitud clave escala logaritmica

Hemos calculado la media armónica entre un tiempo y el siguiente es de 3.47 el cual podemos aproximarlo a 4. De manera que dado un tamaño de X cifras, el tiempo aproximado de ejecución sería de 4^X por lo que para números grandes el tiempo necesario de ejecución sería enorme.

El algoritmo además ocupa una gran cantidad de RAM, ya que para la clave de mayor tamaño ha llegado a ocupar 4.2 Gb.

Conclusión

Este algoritmo no tiene una gran eficiencia a la hora de procesar números grandes ya que si por ejemplo ejecutamos el algoritmo con una clave de 50 cifras tendríamos que $4^{(}50) = 4{,}02e + 19$ años, lo que sería demasiado tiempo para descodificar una clave de ese tamaño.