Práctica 4
Procesos de Ingeniería del Software

Santiago Sañudo Martínez

1. Método de Resolución de la Práctica

La práctica plantea ejercicio el cual consiste en elaborar un tipo de criptación, la cual ya está implementada, repetidas veces, entre 1000 y 10000, de modo que con cada criptación obtenemos cuanto tiempo ha tardado en ms, ya que este tiempo puede variar.

Con estos datos, el tiempo que ha tardado y el número de veces que ha tardado ese tiempo, debemos mostrarlos en un histograma de manera que se pueda mostrar claramente los resultados y observarlos de un solo vistazo.

Para comenzar, hemos creado una interfaz JCriptInterface la cual implementan Jcripta, Jcripto y Jcriptu. La razón de esta interfaz es que será necesario emplearla en la siguiente clase, CalculaTiempos.

La clase CalculaTiempos necesita que se le pase en su constructor un elemento del tipo JCriptInterface y automáticamente comenzara a realizar las iteraciones en bucle, siendo el número de iteraciones el valor definido en la variable estática NUM_EJECUCIONES.

Por cada iteración se realizan 8 criptaciones distintas, y en cada una de ellas se muestra por pantalla que se ha encriptado y cuál es el resultado así como el tiempo que ha tardado pero más importante aún, antes de comenzar la nueva encriptación ejecuta el método addDuracionToMap().

El método addDuracionToMap() es bastante simple, cada duración de una criptacion es la clave de un Map, el cual es un atributo de la clase, Map<Long, Integer> tiempos. Esta clave tiene asociada un valor, el cual será el número de veces que se ha repetido dicho tiempo de encriptación.

El método es simple, si había previamente ya una duración usada como clave igual a la actual en el mapa, modificamos el valor asociado sumándole 1, en caso negativo de no existir, creamos la <clave,valor> con valor 1.

La razón de emplear un mapa en esta situación es que la eficiencia del acceso al conjunto de datos para obtener el número de repeticiones de dicho clave es O(1) usando el mapa, mientras que si usáramos una lista seria de O(n).

Si los datos de muestra no fueran muy grandes no habría problema, pero dado que es un problema estadístico y a mayor número de muestras mejor es el resultado debemos agilizar el acceso a los datos.

Posteriormente de realizar todas las iteraciones ejecuta el método calculaValores().

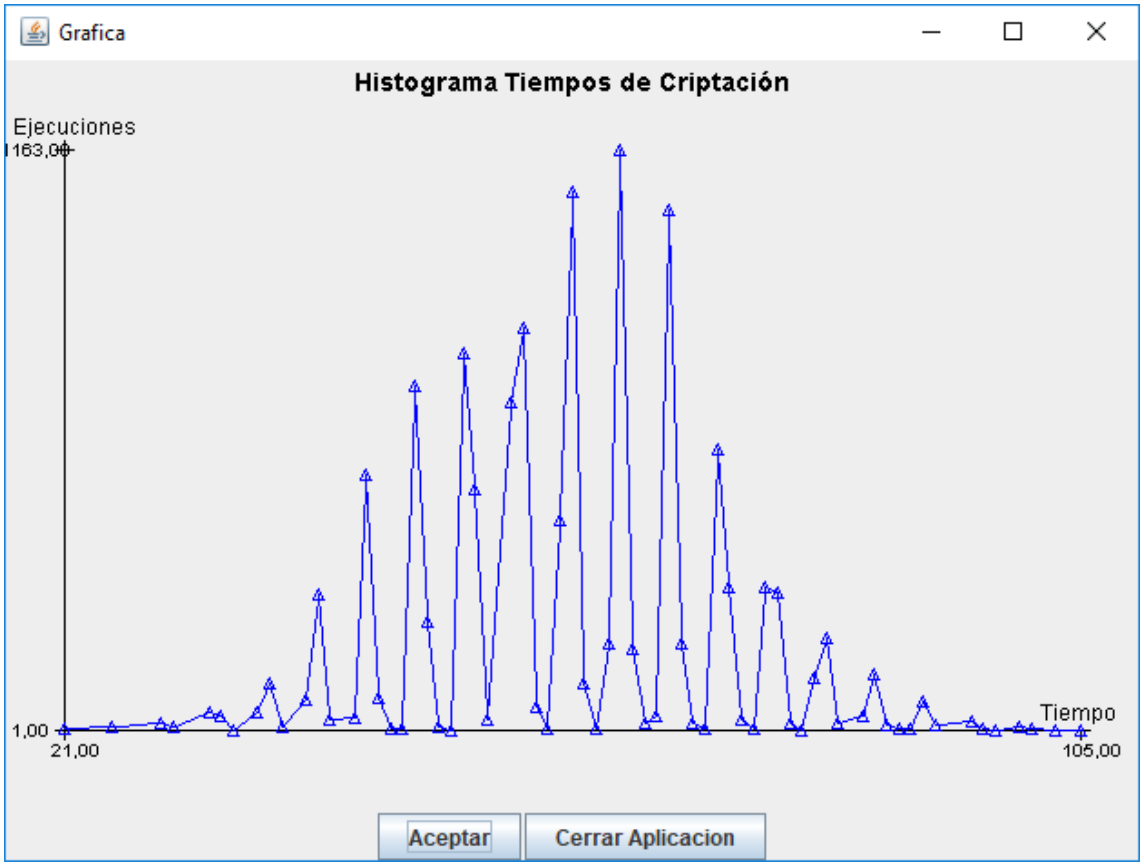
Este método cambia el modelo de datos del mapa a una lista, la cual también es un atributo de la clase, que está formada por el objeto DatoHistograma, el cual posee 2 atributos, tiempoEjecucion y numRepeticiones.

Así mismo, ordena la lista según el tiempo de ejecución de los objetos de menor tiempo de ejecución a mayor tiempo de ejecución.

Posteriormente calcula el mejor tiempo de respuesta, el peor tiempo de respuesta, el tiempo promedio, la desviación estándar y el tiempo de respuesta que está por encima del 99.5% de

los tiempos de respuesta de todas las muestras y los almacena en sus atributos y muestra un mensaje por consola indicando los resultados.

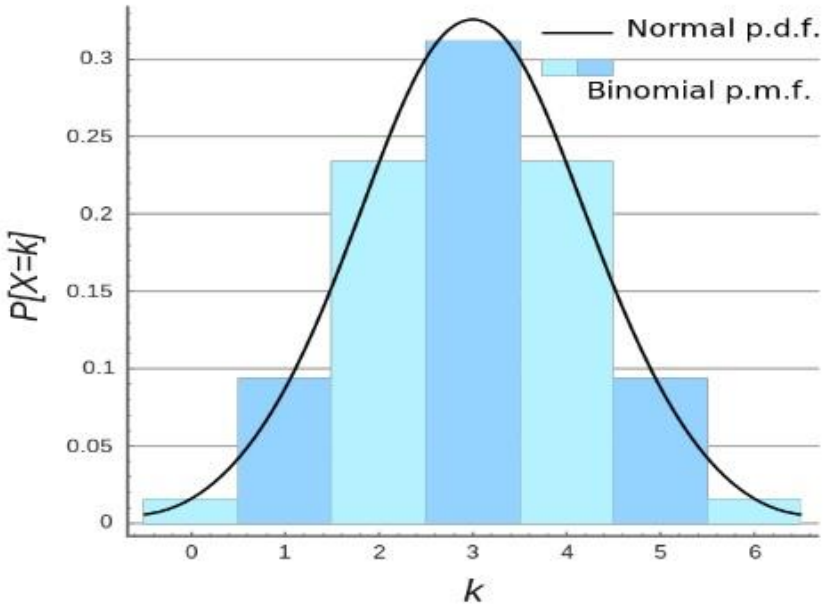
Por último, tenemos la clase Histograma, la cual es el nuevo Main que hemos desarrollado para que genere un objeto de CalculaTiempos empleando el algoritmo de encriptación de Jcripta, y una vez terminado el cálculo muestre todos los resultados en una gráfica la cual tomaría esta forma si usáramos un NUM_ITERACIONES=1250, siendo igual a 10000 encriptaciones:



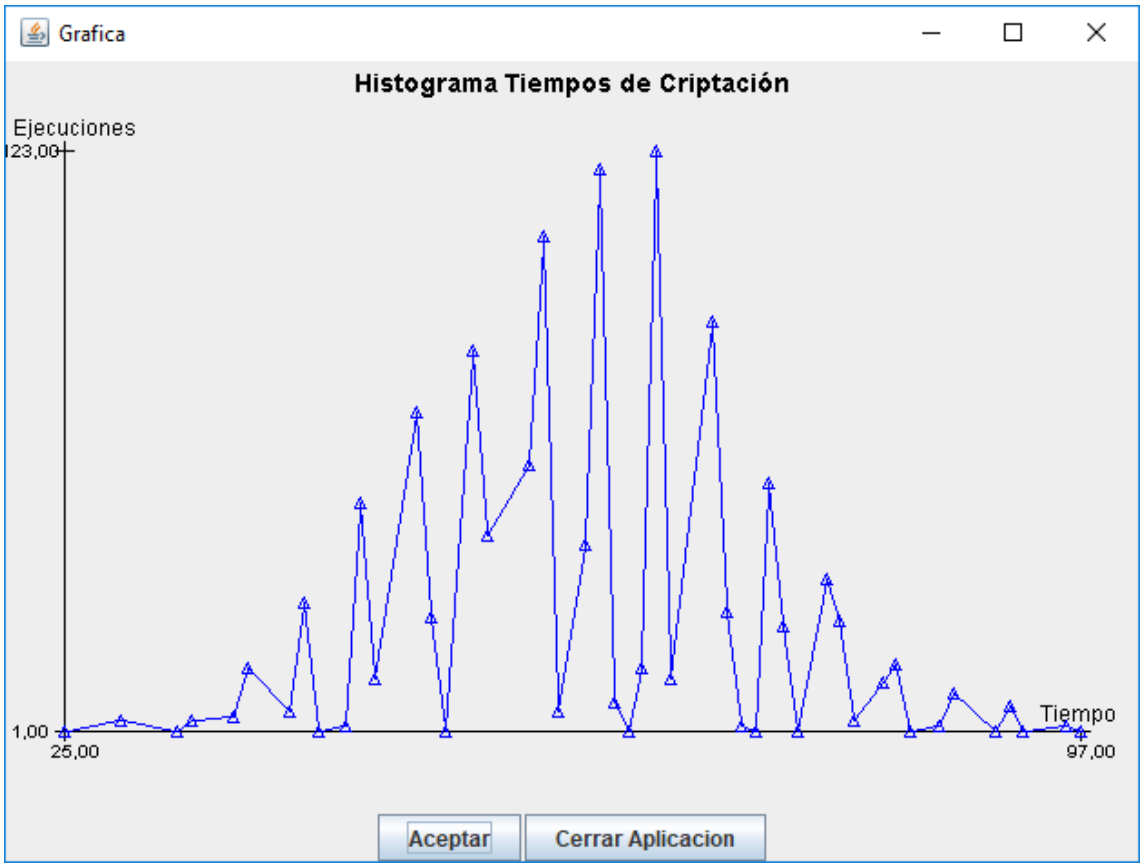
Este tamaño de muestra bastante grande y la realización de esta prueba toma bastante más tiempo de ejecución, sin embargo, los resultados son más precisos gracias a esto.

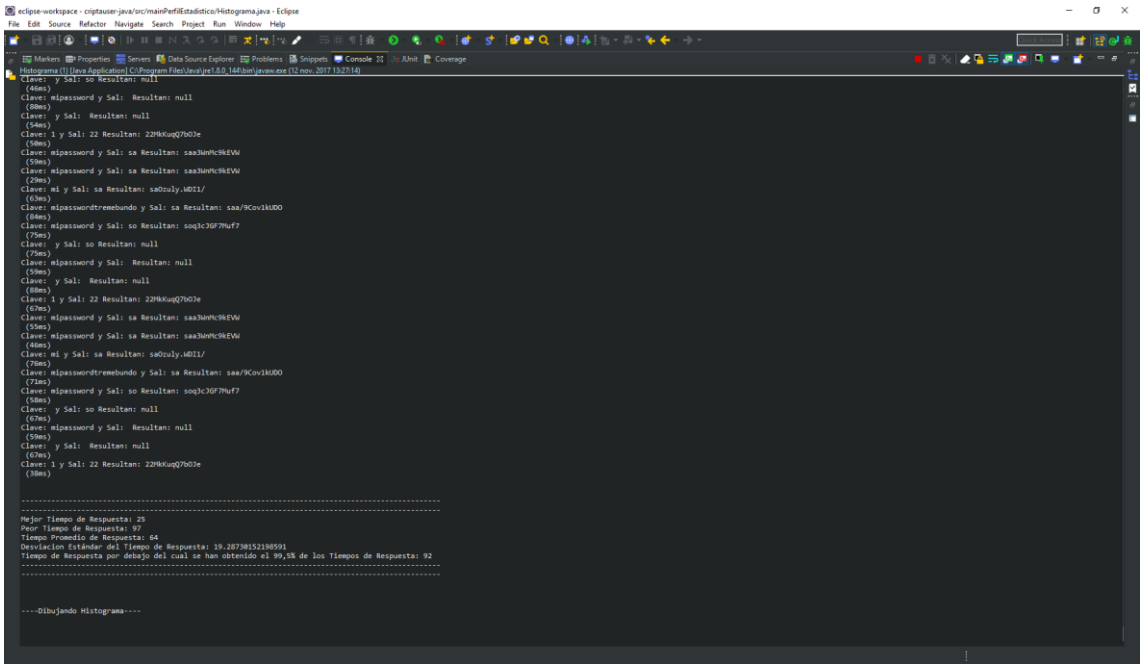
Sin embargo, si usáramos un tamaño de muestras con valor infinito, podríamos observar una con la forma básica de una distribución binomial:

GRÁFICA DE DISTRIBUCIÓN BINOMIAL



Por el contrario, si utilizamos un tamaño de muestra de 2500 elementos podemos observar que la estructura básica sigue siendo similar a la distribución binomial, sin embargo, entre los valores observamos minimos muy dispares, los cuales desaparecerían si aumentamos el tamaño de la muestra como hemos podido comprobar:





2. Resolución de Tests

Para la elaboración de estos hemos desarrollado tests con JUnit en la clase TestCalculaTiempos.

Esta clase antes de comenzar los test calcula los tiempos mencionados previamente, el mejor tiempo de respuesta, el peor tiempo de respuesta, el tiempo promedio, la desviación estándar

y el tiempo de respuesta que está por encima del 99.5% de los tiempos de respuesta de todas las muestras empleando las distintas encriptaciones, Jcripta, Jcripto y Jcriptu.

Posteriormente realiza para cada una de ellas 3 test de modo que compara la desviación de los resultados con un nuevo cálculo de Jcripta, Jcripto y Jcriptu, de modo que los test consisten en 9:

1- Jcripta:
   1.1- Jcripta con Jcripta
   1.2- Jcripta con Jcripto
   1.3- Jcripta con Jcriptu
2- Jcripto:
   2.1- Jcripto con Jcripta
   2.2- Jcripto con Jcripto
   2.3- Jcripto con Jcriptu

3- Jcriptu:
   3.1- Jcriptu con Jcripta
   3.2- Jcriptu con Jcripto
   3.3- Jcriptu con Jcriptu

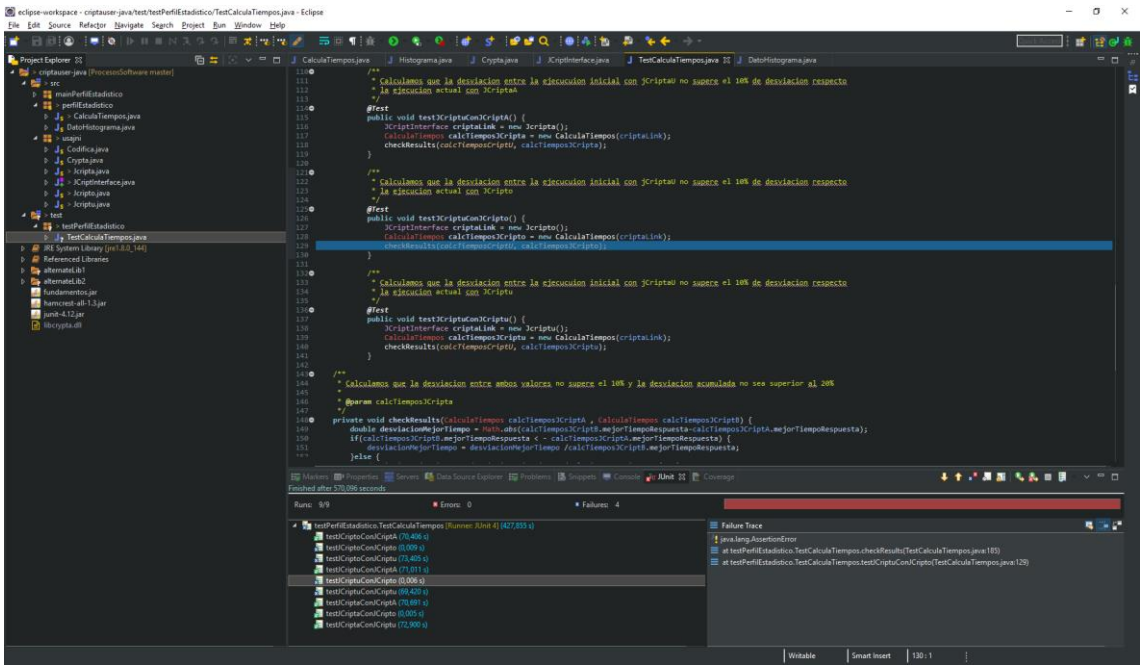
En cada uno de ellos se asegura que los resultados de cada valor clave no posean una desviación superior al 10% y la acumulación de desviaciones de valores claves no supere el 20%, esto se consigue mediante el método checkResults().

3. Informe Resultados Tests

Dado que son muchas pruebas por realizar y cada una toma un tiempo considerable debemos bajar el tiempo, y con ello el tamaño de las muestras, esto puede llevar a resultados inesperados o fallos en los tests, pero si quisiéramos un test perfecto deberíamos tener un tamaño de muestras infinito y por tanto la duración también seria infinita.

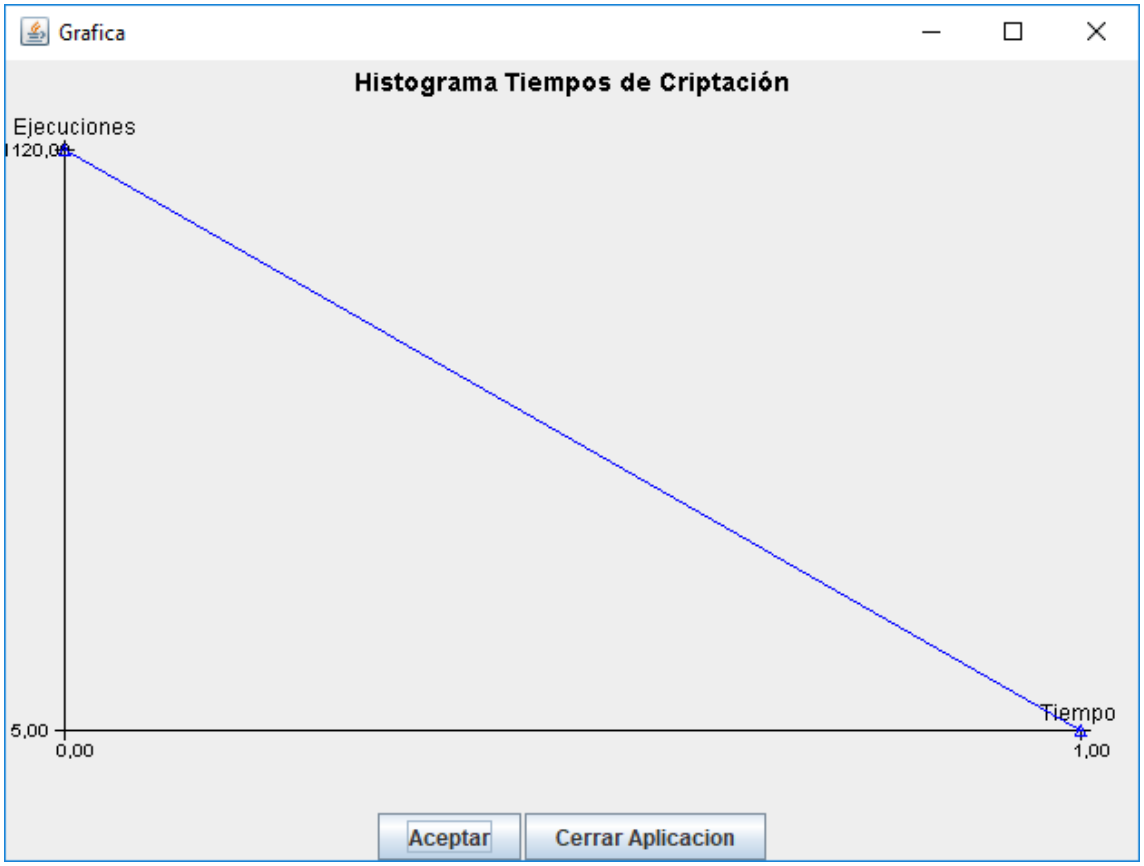
Por ello reducimos el tamaño de muestra a 1000 encriptación, un total de 125 iteraciones.

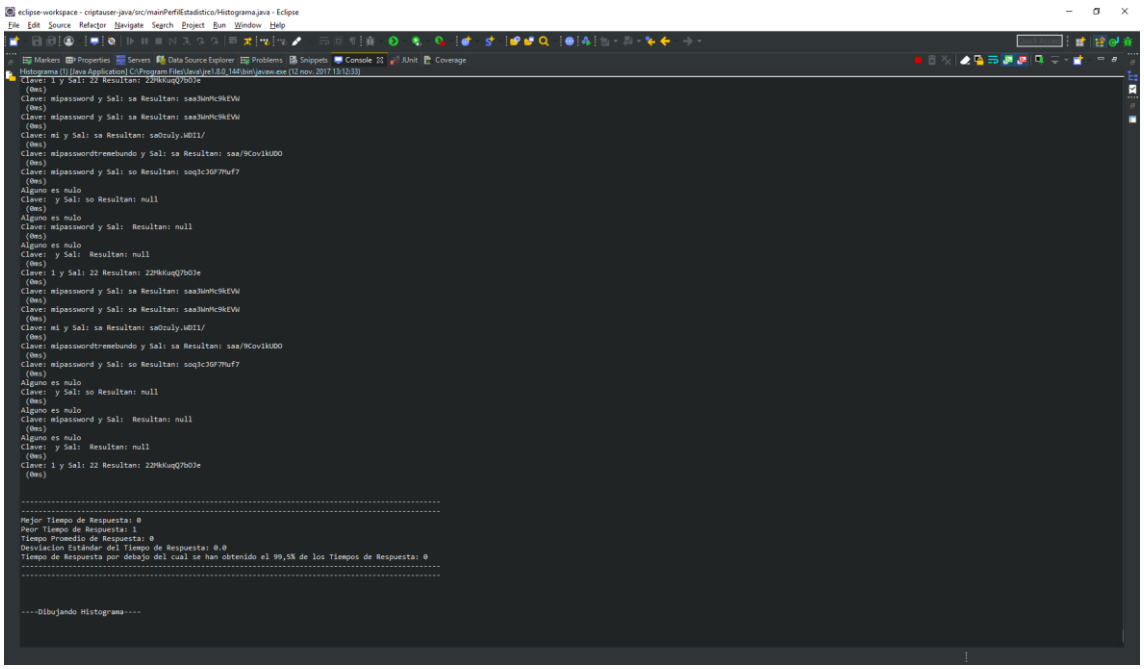
Tras la ejecución de todos los métodos observamos lo siguiente:



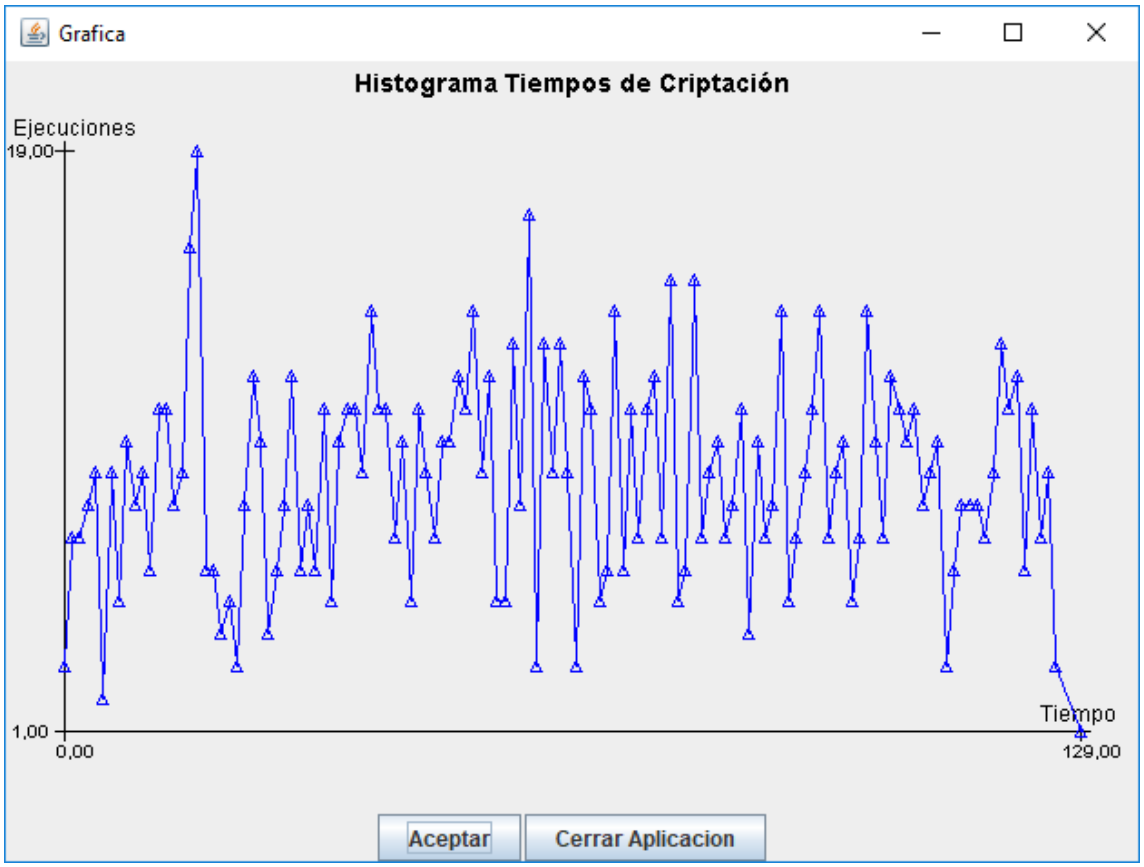
Los test 2.2, 2.3 y 3.2 y 3.3 fallan, si nos centramos un poco más en sus resultados, podemos ver que estos son los diagramas resultantes de realizar la encriptación con dichas claves:

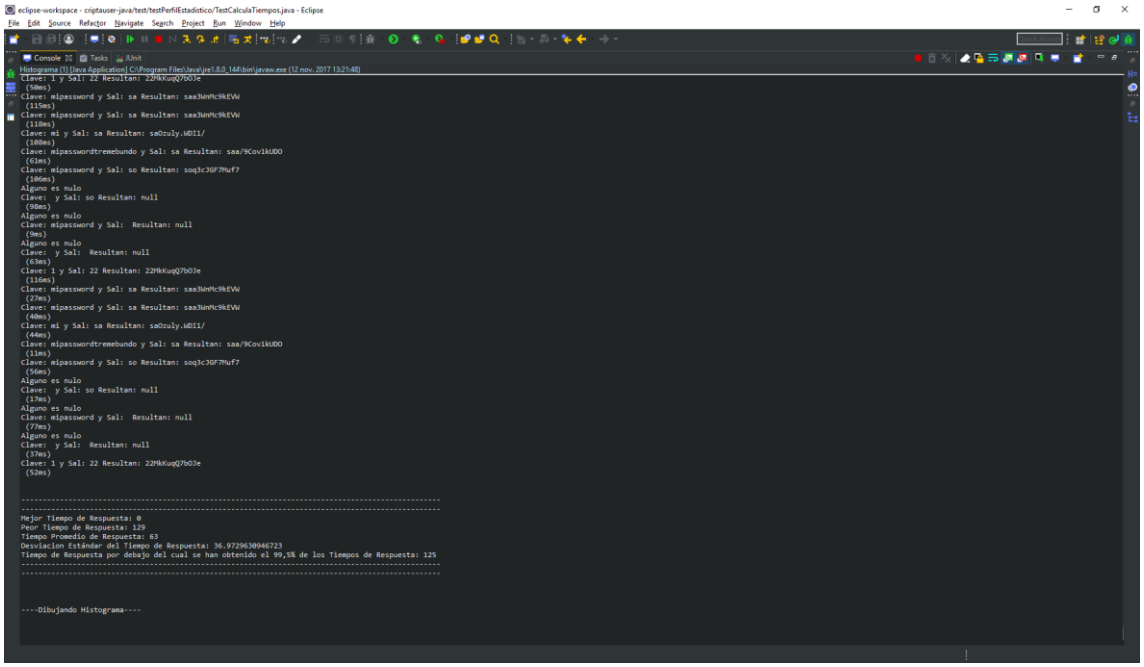
En el caso de JCripto:





En el caso de JCriptu el diagrama toma la siguiente forma:





Como podemos observar son valores muy dispares, por tanto es normal que los tests de Jcripta con las otras criptaciones si pasen debido a que las desviaciones de los valores clave son dentro de las pautas establecidas mientras que estas mismas desviaciones para Jcripto consigo mismo y Jcriptu, asi como Jcriptu consigo mismo y Jcripto no consigan estar dentro de los márgenes de desviación permitidos, 10% individualmente cada valor clave y 20% la desviación acumulada.