Miguel Méndez Murias
UO287687 - Algoritmia

PRÁCTICA 6

Primera parte: La herencia millonaria

En esta parte de la práctica teníamos como objetivo descifrar una serie de claves encriptadas con el algoritmo de encriptación MD5. Estas claves se corresponden a la cantidad de dinero que hay en cada una de las cuentas bancarias (en total 15) de un pariente que ha fallecido y que nos ha dejado estas cuentas y las claves encriptadas para poder acceder a ellas.
Sabemos además que las claves tienen una cantidad de caracteres que varía entre tres y cincuenta y que cada carácter es cualquier letra mayúscula, excepto la "Ñ".
Para descifrar las claves, tenemos que desarrollar una clase que permita descifrar las claves a partir del código MD5.

¿Cómo funciona el algoritmo MD5?

El algoritmo MD5 se basa en una función hash que convierte los datos en una cadena de 32 caracteres, independientemente del tamaño del dato a encriptar original. Aunque actualmente se usa principalmente para la autenticación, es decir, para comprobar que un archivo recibido coincide con el original, en nuestro caso se utiliza para el cifrado de datos.

Desarrollo del algoritmo para desencriptar las claves

Ya que no conocemos ninguna forma de descodificar una clave encriptada en MD5, la única solución para poder descifrar las claves es mediante backtracking.
Para ello, primero leemos las claves encriptadas del fichero correspondiente y almacenamos los valores que puede tener cada carácter de la clave original en una lista.
Posteriormente, procedemos a hacer un método recursivo que implemente backtracking, de forma que nos imprima por pantalla la clave descodificada y el tiempo que tardó en descodificarla si lo ha conseguido. Además, limitamos el tamaño máximo que pueden tener las claves, ya que en el peor caso posible, es decir con 50 caracteres y próximos al final, el tiempo de ejecución sería demasiado grande y no estaríamos vivos para obtener las claves descifradas. Por ello, realizamos backtracking mientras que el nivel de profundidad actual sea menor que el nivel máximo indicado.

Tiempos de desencriptado (ms)

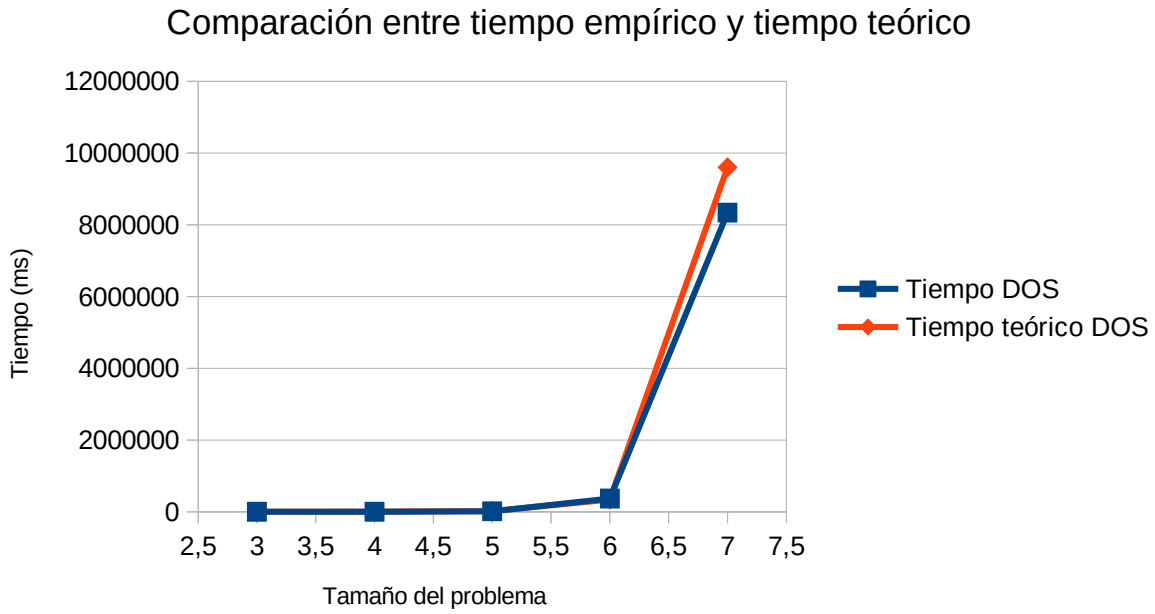
Tamaño de palabra máximo	Clave : DOS	Clave : SEIS	Clave : NUEVE	Clave : QUINCE
3	131	-	-	-
4	764	2635	-	-
5	13070	60703	46341	-
6	369354	1566576	1237276	1462277
7	8341026	38552139	29516961	3570219

Miguel Méndez Murias
UO287687 - Algoritmia

Para comprobar que se cumple la complejidad esperada, es decir, $O(26^n)$, en el que la n corresponde al tamaño de palabra máximo y el 26 a la cantidad de posibles valores que puede tomar cada carácter, comparamos el tiempo que tarda el algoritmo en desencriptar la misma clave según aumentamos el tamaño de la palabra máximo.

En este caso, elegimos la clave “DOS” para comprobar la complejidad:

Tamaño de palabra máximo	Tiempo(ms): DOS	Tiempo teórico(ms): DOS
3	131	131
4	764	3406
5	13070	19864
6	369354	339820
7	8341026	9603204



Como se puede comprobar tanto en los tiempos como en la gráfica anterior, se cumple la complejidad esperada del algoritmo.

*Nota: para calcular el tiempo teórico utilizamos la siguiente fórmula:

$$t2 = \frac{f(n_2)}{f(n_1)} * t1$$ siendo $f(n) = 26^n$

Otras formas de realizar el descifrado de claves

En este caso, como sabemos que cada clave corresponde a un número escrito en letras mayúsculas, podríamos realizar un diccionario que contuviese una gran cantidad de números en este formato e ir comprobando si el código MD5 de cada fila del diccionario coincide con alguna clave, de forma que podríamos descifrar las claves de una forma más

óptima que con backtracking, ya que podríamos evitar una gran cantidad de combinaciones y reducir la complejidad.
Las claves que se obtuvieron con este método fueron las siguientes:

- DOS
- SEIS
- NUEVE
- QUINCE
- DIECISEIS
- VEINTITRES
- TREINTATRES
- CIENTOSESENTAYDOS
- MILCIENTOVEINTICUATRO
- MILDOSCIENTOSVEINTITRES
- DOSMILDOSCIENTOSVEINTIUNO
- VEINTITRESMILDOSCIENTOSCATORCE
- CIENTOVEINTIOCHOMILDOSCIENTOSTRES

Si bien es cierto que solo se han descifrado trece claves y no quince como pide el ejercicio, esto se debe a que el tamaño del diccionario era ya muy grande y no generamos más números. Sin embargo, si que sería posible continuar con este método, y tal y como hemos comprobado, se conseguiría descifrar las claves de una manera mucho más rápida que con backtracking.

Miguel Méndez Murias
UO287687 - Algoritmia

Segunda parte: El viajante de comercio

En esta parte de la práctica teníamos que averiguar cual era el camino de menor coste que podía seguir un viajante para, saliendo de una ciudad, visitar todas las otras volviendo finalmente a la ciudad de origen (sin repetir ninguna ciudad), es decir, teníamos que hallar el ciclo hamiltoniano.
El problema estaba modelado como un grafo completo y dirigido, en el que cada arista tenía un peso positivo.

Resolución del problema

Para resolver este problema, implementamos un algoritmo utilizando backtracking de forma que partiendo de la ciudad inicial, se recorriesen todas las ciudades solo una vez para volver finalmente a la ciudad inicial. Para ello, teníamos que almacenar los nodos visitados y si el nivel de profundidad coincidía con el buscado (es decir, se recorrieron todas las ciudades), se comprobaba si el coste de esa solución era menor que el de la mejor solución encontrada hasta el momento, y en este caso se sustituía. En caso de que no se recorriesen todas las ciudades, se busca una nueva ciudad que no haya sido visitada y se llama a este algoritmo de forma recursiva.

Pruebas

Para comprobar el correcto funcionamiento del algoritmo programado disponíamos de unos ficheros con un número de ciudades diferentes y que representaban un grafo diferente. Además, sabíamos cual era el camino de coste mínimo que se tenía que seguir para obtener el ciclo hamiltoniano buscado, así como el coste del mismo. Sabiendo esto, para comprobar el funcionamiento, ejecutamos el programa pasando como entrada cada uno de esos grafos y comprobamos que la salida obtenida se correspondía con la esperada. Como ambas eran iguales, podemos garantizar el buen funcionamiento de nuestro algoritmo.

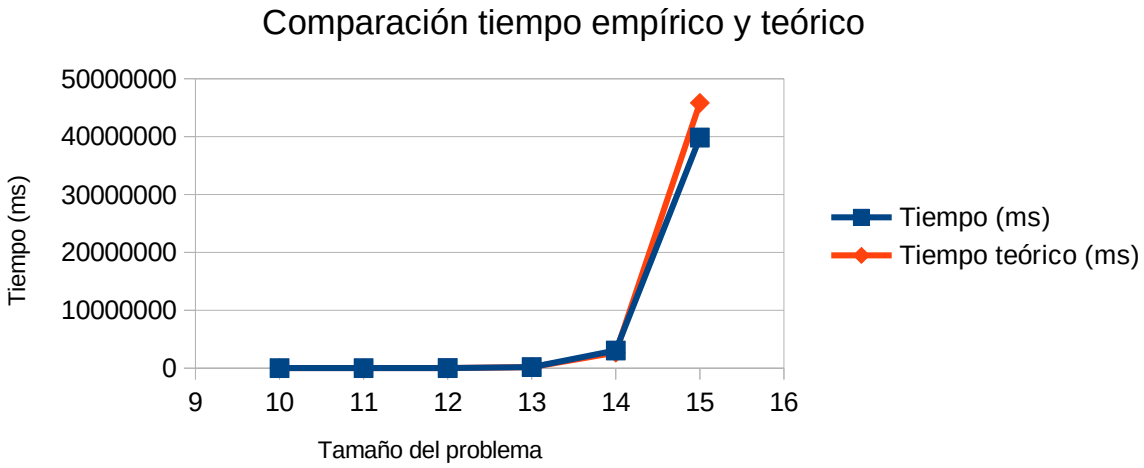
Medición de tiempos

Para realizar la medición de tiempos, creamos una nueva clase llamada ViajanteTiemposBK, la cual generaba matrices cuadradas de distintos tamaños que representaban el grafo descrito anteriormente y de esta forma podíamos medir el tiempo de ejecución del algoritmo para distintos tamaños del problema.
Además, sabemos que el algoritmo utilizado debería tener una complejidad O((n-1)!) por lo que también calculamos cuales deberían seres los tiempos teóricos necesarios para cada tamaño del problema.

Los resultados fueron los siguientes:

Tamaño del problema	Tiempo (ms)	Tiempo teórico (ms)
10	127	127
11	1263	1397
12	13308	15156
13	191417	173004

14	3054637	2679838
15	39840719	45819555



Como se puede observar tanto en la tabla de tiempos como en la gráfica, se cumple la complejidad esperada del algoritmo desarrollado.

*Nota: para calcular el tiempo teórico utilizamos la siguiente fórmula:

$$t2 = \frac{f(n_2)}{f(n_1)} * t1$$ siendo *f(n) = (n-1)!*