Estructuras de Datos y Algoritmos

Práctica I - Curso 2016/17

Ataque Inminente (I)

Introducción

Tras abandonar los estudios de informática al no poder aprobar la asignatura de Estructuras de Datos, encontraste trabajo en un Grupo de Operaciones Especiales de las fuerzas del orden de tu Comunidad Autónoma, donde llevas una vida tranquila y feliz. Acabáis de asaltar una casa donde se encontraba un peligroso extremista que iba a llevar a cabo un ataque inminente, pero desafortunadamente ha conseguido huir. Al registrar la casa habéis encontrado el ordenador portátil que usaba para contactar con su organización, Evil Corp. Esta comunicación se llevaba a cabo mediante correo electrónico en formato texto plano y sin cifrar (no son unos delincuentes muy sofisticados), y el portátil no tiene contraseña, por lo que al ser el único miembro del grupo con algún conocimiento de informática tus compañeros te están presionando para que accedas a esos correos electrónicos y podáis averiguar cuáles eran sus planes. El tiempo apremia porque es seguro que el extremista huido va a llevar a cabo su ataque lo antes posible.

Por desgracia al abrir el gestor de correo descubres que no hay ninguno dirigido a la dirección de su organización, **evil.corp@mad.org** (todos estos destalles los conocéis por un agente infiltrado). Seguramente antes de huir el extremista ha tenido tiempo de borrar esos correos (y borrarlos también de la carpeta de eliminados).

Tus conocimientos te permiten suponer que aunque los mensajes hayan sido borrados es probable que todavía se encuentren en el fichero de datos del gestor de correo, ya que en general estos programas siguen una *estrategia perezosa* que consiste en que al borrar un elemento se marca el espacio que ocupaba como libre pero no se reestructura el fichero ni se sobreescribe inmediatamente esa información. En teoría es factible acceder a esos mensajes haciendo una **búsqueda** de la cadena "evil.corp@mad.org" sobre el fichero de datos. Existen ciertas complicaciones, sin embargo, que salen a la luz cuando miras en internet la descripción de la estructura de los ficheros de datos (https://msdn.microsoft.com/en-us/library/ff385210(v=office.12).aspx) :

- El gestor de correo usa **ofuscación de datos** para almacenar los mensajes en el fichero: Para cada correo se genera al azar una clave (un número entero entre 0 y 65535) que se utiliza para sustituir cada carácter del mensaje por otro siguiendo un determinado algoritmo (os lo describo más adelante).
- La clave asociada a cada mensaje se **pierde** al ser borrado.

Por tanto la única estrategia viable parece ser la de des-ofuscar los datos 65536 veces, una vez por cada posible clave, y buscar la cadena sobre esos datos para comprobar si alguno de los mensajes se ofuscó con esa clave en particular.

Para más inri el fichero de datos de correo ocupa varios gigas (a pesar de ser extremista su vida social era bastante intensa) y seguramente no se pueda volcar a memoria directamente sino que deba ser procesado en trozos.

Objetivos de la práctica

Se debe implementar una aplicación a la que se indique el nombre del fichero de datos y la cadena de texto que debe buscar y que procese los datos de la manera descrita anteriormente: Se des-ofuscan los datos para cada posible valor de la clave (65536 veces) y en cada una de esas iteraciones se busca sobre los datos des-ofuscados la cadena de texto. Para cada concordancia se mostrará en pantalla los 100 caracteres anteriores y los 500 posteriores a ella (se supone que los mensajes no son muy largos)

El objetivo es averiguar si con éste método nuestro protagonista podrá obtener los mensajes en **un tiempo razonable** o bien es necesario que cambie de estrategia. En la página de la asignatura se proporcionarán ficheros de prueba de tamaños crecientes para que podáis comprobar tanto la corrección de vuestra aplicación como la forma en que se incrementa el tiempo necesario para resolver el problema.

Detalles Técnicos

La ofuscación y des-ofuscación son simétricas (al volver a aplicar el algoritmo con la misma clave a un texto ofuscado se obtiene el texto original) por lo que sólo se usa un algoritmo, que se describe a continuación en pseudocódigo:

```
# txt es el vector con los datos (bytes) que se desean (des)ofuscar
# clave es un entero entre 0 y 65535
# vPR, vPS y vPI son vectores constantes predefinidos
función ofuscar(txt, clave):
   for i = 0..length(txt)-1:
        w0 = clave mod 256
        w1 = clave div 256
        b = txt[i];  b = (b+w0) mod 256
        b = vPR[b];  b = (b+w1) mod 256
        b = vPS[b];  b = (b-w1+256) mod 256
        b = vPI[b];  b = (b-w0+256) mod 256
        txt[i] = b
        clave = (clave+1) mod 65536
```

Los vectores predefinidos vPR, vPS y vPI tienen índices de 0 a 255 y almacenan enteros en el rango de 0 a 255 (es decir traducen bytes en bytes). Su contenido es el siguiente:

```
  # Permutación directa
  vPR = [
    65,54,19,98,168,33,110,187,244,22,204,4,127,100,232,93,
    30,242,203,42,116,197,94,53,210,149,71,158,150,45,154,136,
    76,125,132,63,219,172,49,182,72,95,246,196,216,57,139,231,
    35,59,56,142,200,193,223,37,177,32,165,70,96,78,156,251,
    170,211,86,81,69,124,85,0,7,201,43,157,133,155,9,160,
    143,173,179,15,99,171,137,75,215,167,21,90,113,102,66,191,
    38,74,107,152,250,234,119,83,178,112,5,44,253,89,58,134,
```

```
  126,206,6,235,130,120,87,199,141,67,175,180,28,212,91,205,
  226,233,39,79,195,8,114,128,207,176,239,245,40,109,190,48,
  77,52,146,213,14,60,34,50,229,228,249,159,194,209,10,129,
  18,225,238,145,131,118,227,151,230,97,138,23,121,164,183,220,
  144,122,92,140,2,166,202,105,222,80,26,17,147,185,82,135,
  88,252,237,29,55,73,27,106,224,41,51,153,189,108,217,148,
  243,64,84,111,240,198,115,184,214,62,101,24,68,31,221,103,
  16,241,12,25,236,174,3,161,20,123,169,11,255,248,163,192,
  162,1,247,46,188,36,104,117,13,254,186,47,181,208,218,61]
```

```
# Permutación inversa vPR[vPI[i]] = i, vPI[vPR[i]] = i
vPI = [
  71,241,180,230,11,106,114,72,133,78,158,235,226,248,148,83,
  224,187,160,2,232,90,9,171,219,227,186,198,124,195,16,221,
  57,5,150,48,245,55,96,130,140,201,19,74,107,29,243,251,
  143,38,151,202,145,23,1,196,50,45,110,49,149,255,217,35,
  209,0,94,121,220,68,59,26,40,197,97,87,32,144,61,131,
  185,67,190,103,210,70,66,118,192,109,91,126,178,15,22,41,
  60,169,3,84,13,218,93,223,246,183,199,98,205,141,6,211,
  105,92,134,214,20,247,165,102,117,172,177,233,69,33,112,12,
  135,159,116,164,34,76,111,191,31,86,170,46,179,120,51,80,
  176,163,146,188,207,25,28,167,99,203,30,77,62,75,27,155,
  79,231,240,238,173,58,181,89,4,234,64,85,37,81,229,122,
  137,56,104,82,123,252,39,174,215,189,250,7,244,204,142,95,
  239,53,156,132,43,21,213,119,52,73,182,18,10,127,113,136,
  253,157,24,65,125,147,216,88,44,206,254,36,175,222,184,54,
  200,161,128,166,153,152,168,47,14,129,101,115,228,194,162,138,
  212,225,17,208,8,139,42,242,237,154,100,63,193,108,249,236]
```

```
# Autopermutación vPS[vPS[i]] = i
vPS = [
  20,83,15,86,179,200,122,156,235,101,72,23,22,21,159,2,
  204,84,124,131,0,13,12,11,162,98,168,118,219,217,237,199,
  197,164,220,172,133,116,214,208,167,155,174,154,150,113,102,195,
  99,153,184,221,115,146,142,132,125,165,94,209,93,147,177,87,
  81,80,128,137,82,148,79,78,10,107,188,141,127,110,71,70,
  65,64,68,1,17,203,3,63,247,244,225,169,143,60,58,249,
  251,240,25,48,130,9,46,201,157,160,134,73,238,111,77,109,
  196,45,129,52,37,135,27,136,170,252,6,161,18,56,253,76,
  66,114,100,19,55,36,106,117,119,67,255,230,180,75,54,92,
  228,216,53,61,69,185,44,236,183,49,43,41,7,104,163,14,
  105,123,24,158,33,57,190,40,26,91,120,245,35,202,42,176,
  175,62,254,4,140,231,229,152,50,149,211,246,74,232,166,234,
  233,243,213,47,112,32,242,31,5,103,173,85,16,206,205,227,
  39,59,218,186,215,194,38,212,145,29,210,28,34,51,248,250,
  241,90,239,207,144,182,139,181,189,192,191,8,151,30,108,226,
  97,224,198,193,89,171,187,88,222,95,223,96,121,126,178,138]
```

En los siguientes enlaces podéis encontrar una descripción alternativa (en lenguaje C) del algoritmo (https://msdn.microsoft.com/en-us/library/ff386960(v=office.12).aspx) y las tablas anteriores (https://msdn.microsoft.com/en-us/library/ff386229(v=office.12).aspx) .

Como algoritmo de búsqueda se recomienda usar primero el algoritmo "clásico" y (opcional, sólo si se desea mejorar la calificación) comprobar si Rabin-Karp proporciona una mejora apreciable. **Nota:** No se pueden usar funciones predefinidas de búsqueda de texto que puedan existir en el entorno de programación, debéis programarlas vosotros.

Presentación y Evaluación de la práctica

Se pueden utilizar los lenguajes Java, Python, C, Pascal, Haskell o R para la implementación de la aplicación. Si se desea utilizar otro lenguaje debe obtenerse primero la autorización del profesor.

Se recomienda que la práctica se realice en parejas de dos personas, aunque en casos especiales se puede permitir la realización individual o en grupos de 3 personas (en este último caso la nota obtenida sufre una minoración)

Para una correcta evaluación de la práctica el alumno deberá:

1. Presentar electrónicamente (por el Aula Virtual de la Escuela o por correo electrónico), antes de las 23:59 del 16 de octubre de 2016, un fichero comprimido que contenga el código fuente de la aplicación utilizada para resolver el problema planteado. En el código fuente debe aparecer (como comentario) el nombre de quienes han realizado la práctica.
2. Presentarse a la sesión de evaluación que le corresponda según su grupo de laboratorio en la semana del 17 al 20 de octubre de 2016. En esta sesión se probará la aplicación realizada (con nuevos datos) y se presentará la estimación del tiempo necesario para resolver el problema real. Es posible que en la sesión se solicite la modificación del código de la aplicación.

En el caso de realización por parejas (la situación habitual), tan sólo es necesario que uno cualquiera de ellos realice la presentación electrónica. En la evaluación, sin embargo, si es necesaria la presencia de ambos y la evaluación puede ser distinta para cada uno de ellos.