PRÁCTICA 01

La práctica corresponde al
Tema 2: Complejidad computacional

Peso Total en la nota: ***6%***

Objetivos

Se espera que el alumno se familiarice con el concepto de complejidad computacional[1]. Para ello, el ejercicio consiste en su análisis y cálculo teórico de algoritmos de ordenación, así como su comprobación práctica del consumo de recursos.
Se comprobará como, a pesar de poder darse diferentes algoritmos que solucionan correctamente un mismo problema, esta característica de los algoritmos hace claramente ventajosa a unas soluciones frente a otras. Este conocimiento se aplicará a lo largo del resto de la asignatura, permitiendo comparar las ventajas de las diferentes Estructuras de Datos.
Adicionalmente se trabajará con la Genericidad de Java, que permite definir los tipos de datos a almacenar en una colección. Es una característica muy importante que mantienen todos los componentes del *Collections Framework*.

Dinámica

Se pretende que el alumno se familiarice con el concepto de Eficiencia Algorítmica[1] implementando diferentes métodos de consulta y procesamiento sobre conjuntos de datos de diferentes tamaños. El objetivo final del ejercicio es poder completar un informe en el que se incluyan los resultados de los experimentos realizados por el alumno, así como las conclusiones que se alcanzan a la vista de los mismos.
Por la naturaleza de este ejercicio, se debe tener en cuenta que en esta práctica se valorará de forma especial el informe presentado (además del código fuente del programa).

Enunciado de la Práctica

Deseamos localizar el mismo subconjunto de objetos localizados en posiciones contiguas dentro de dos vectores diferentes de objetos. No se debe confundir con el problema de las diferentes sub-secuencias comunes a dos colecciones de objetos. En ese caso, se busca solo aquella que sea la de mayor longitud.

Por ejemplo, supongamos los vectores:

[**1,2,4,5,3,4,6,7,9,4,5,33,45,63,2,4**,44,32]

[4,2,**1,2,4,5,3,4,6,7,9,4,5,33,45,63,2,4**,55,2,9,10]

En este caso, podemos solucionarlo a simple vista y obtener como resultado: [1,2,4,5,3,4,6,7,9,4,5,33,45,63,2,4]

Para simplificar el ejercicio, no se considerará el buscar si existen más vectores solución con el mismo tamaño que la solución aportada: solamente hay que devolver la última secuencia de mayor tamaño encontrada en ambos vectores.

[1] https://es.wikipedia.org/wiki/Eficiencia_Algor%C3%ADtmica

Advertencia: La clase deberá funcionar con cualquier tipo de datos (números, caracteres, cadenas...)

Aproximaciones al problema

1) Aproximación Iterativa

La aproximación más directa que comúnmente se podría considerar es la puramente iterativa. En ella, podemos empezar por extraer todas las sub-colecciones de elementos contiguos de tamaño *1, 2 ... n* y comprobar si está presente en cualquier posición de la segunda. En tal caso iremos comprobando si es de mayor tamaño a la localizada pasos anteriores y nos quedaremos con la más larga.
Podemos considerar el siguiente algoritmo:

```
SubArrayComunMasLargo_metodo1(C1, C2)
Entradas:
       C1 [1 . . . n]// Primer vector de elementos a analizar (tamaño n)
       C2 [1 . . . m]// Segundo vector de elementos a analizar (tamaño m)
Salida:
       R[1..x]            // Vector respuesta

maxLongitud = 0
inicioSecuencia1 = 0
inicioSecuencia2 = 0
tamProbado = 0
tamSecuencia1 = tamanio(C1)
tamSecuencia2 = tamanio(C2)

listaResultado = null;

PARA inicioSecuencia1 = 0 HASTA tamSecuencia1
  PARA tamProbado = 1 HASTA (tamSecuencia1-inicioSecuencia1)
    PARA inicioSecuencia2 = 0 HASTA (tamSecuencia2–tamProbado)
     SI(C1.subList(inicioSecuencia1,inicioSecuencia1+tamProbado) ==
C2.subList(inicioSecuencia2,inicioSecuencia2+tamProbado))
        ENTONCES
          SI maxLongitud < tamProbado ENTONCES
            R=C1.subList(inicioSecuencia1,inicioSecuencia1+tamProbado)
            maxLongitud = tamProbado
        FIN-SI
     FIN-SI
    SIGUIENTE
  SIGUIENTE
SIGUIENTE
DEVOLVER R
```

Se permite al alumno incluir pequeñas modificaciones al algoritmo básico que puede disminuir ligeramente la complejidad algorítmica del mismo. En caso de realizarlas, se pide que se incluya un razonamiento de las modificaciones en el informe asociado a la entrega.

2) Aproximación Mediante Programación Dinámica

En el segundo caso, se solicita implementar la solución al mismo problema, la “Sub-secuencia Común más Larga”, pero empleando el paradigma de la programación dinámica[2].

[2] https://es.wikipedia.org/wiki/Programaci%C3%B3n_din%C3%A1mica

Para la resolución en este caso, se crea una matriz de enteros de tantas filas como elementos tiene la primera colección más uno y tantas columnas como los de la segunda más otra unidad. La primera fila y la primera columna se inicializan a 0.
Para cumplimentar cada celda determinada por las coordenadas "i" y "j", se comparan el elemento (i-ésimo -1) de la colección 1 con el (j-ésimo -1) de la colección 2:

- Si son distintos se guarda un cero en esas coordenadas.
- Si son iguales, se guarda el valor obtenido de la posición [i-1,j-1] de la tabla de cálculo más una unidad.

El valor almacenado indica la longitud de subconjunto contiguo común más largo encontrado hasta ese instante y que termina en la posición "i" de la colección 1 o en el "j" de la colección 2.
Para una aproximación más sencilla, no estudiaremos si hay más de un subconjunto de la misma longitud común a las dos tablas, por lo que iremos almacenando los elementos mayores para su devolución al concluir la búsqueda.

		4	2	1	2	4	5	3	4	6	7	9	4	5	33	45	63	2	4	55	2	9	10
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
4	0	1	0	0	0	3	0	0	1	0	0	0	1	0	0	0	0	0	2	0	0	0	0
5	0	0	0	0	0	0	4	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	1	0	0	6	0	0	0	1	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	1	0
4	0	1	0	0	0	1	0	0	1	0	0	0	10	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	2	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0
63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	1	0	0
4	0	1	0	0	0	2	0	0	1	0	0	0	1	0	0	0	0	0	16	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Un pseudocódigo que nos permitiría obtener la solución podría ser el siguiente:

```
SubArrayComunMasLargo_metodo2()
Entradas:
C1 [0 . . . n]       // Vector Colección1 de elementos a analizar
C2 [0 . . . m]       // Vector Colección2 de elementos a analizar
Salida:
Resultado            // Vector resultado
MatrizCalculo[0..n+1][0..n+1]
PARA i=0 HASTA n+1
 PARA j=0 HASTA m+1
  // A partir de aquí, construimos la tabla
  SI i=0 O j=0 ENTONCES
    //Inicializamos la primera fila y la primera columna con 0
    MatrizCalculo[i][j]=0
   SI NO SI C1[i-1] = C2 [j-1] ENTONCES
     // Si son iguales incrementamos el número de caracteres consecutivos
      MatrizCalculo[i][j] = MatrizCalculo[i-1][j-1]+1
     SI MatrizCalculo[i][j] > MaxLongitud ENTONCES
```

Esta técnica se estudiará de forma extensa en la asignatura de "Algoritmia"

```
        // Si hemos encontrado un máximo inicializamos el valor de la longitud
        máxima y almacenamos en  resultado
        MaxLongitud = MatrizCalculo[i][j]
        Resultado = C1[i-MaxLongitud..i]
      SI NO:
        // si no son iguales se inserta 0
        MatrizCalculo[i][j]=0
      FIN SI
    FIN SI
  FIN SI
DEVOLVER Resultado
```

Simplificaciones

- Si existen más de una serie común solamente se devuelve la última que aparezca.

Análisis

Tras realizar la implementación de los métodos, se pide completar un informe en el que se incluya por cada uno de los nuevos métodos:

- descripción textual del funcionamiento del mismo
- razonamiento, aunque sea textual, de su complejidad algorítmica (O grande)

Este análisis teórico se deberá completar con un análisis empírico de funcionamiento, para comprobar si se cumplen los razonamientos teóricos. Para ello, se pide a los alumnos que creen una clase de prueba que permita medir el tiempo en que se completa cada uno de los métodos, según vaya variando el número de elementos que se encuentran almacenados en la colección.

Para ello se facilita la clase de apoyo `GeneradorEnteros`, que permite generar de forma automática, los elementos con los que poder rellenar la colección, especificando simplemente el número de elementos que se desean almacenar.

Se puede ver varios ejemplos de problemas que incluyen test similares en los ejemplos facilitados como ayuda en la asignatura. Se sugiere al alumno emplear ésta como modelo para completar la suya. Es interesante destacar que, en el caso del ejemplo, los métodos a los que se llaman están en la misma clase que realiza la prueba por cuestiones de simplificación. En el caso de la práctica, es más recomendable separar estos métodos en otra clase.

El resultado de este análisis es la obtención de un listado de tiempos de proceso que se puede representar en una o varias gráficas (se pueden copiar fácilmente a un software de hoja de cálculo, por ejemplo) y que permita comprobar de forma visual la tendencia del algoritmo.

Condiciones de entrega

- La práctica se realizará en grupos de 1 o 2 personas.
- La entrega ser hará SOLAMENTE por medio de la plataforma UBUVirtual. Cada miembro del grupo deberá subir su propia copia de la solución. No se admitirán soluciones fuera de plazo.
- La entrega incluirá la información de quienes son los autores de la misma, tanto en los ficheros de código fuente como en los documentos asociados.

- Cada entrega consistirá en un fichero comprimido (formato .zip), con la estructura de nombre "`Apellidos1Nombre1_Apellidos2Nombre2`".
- Deberá que incluir comprimido:
  - Código fuente de la solución (dentro de la estructura de paquetes necesaria)
  - Informe de Complejidad algorítmica. En formato PDF.
  - Documentación generada en javadoc
  - No hace falta entregar ficheros binarios

Informe

En el caso de la presente práctica se debe incluir un informe adicional. Ver el contenido solicitado en la sección "**Análisis"**.

Criterios de Evaluación

Dada la naturaleza y objetivo de la práctica, la documentación, el análisis de la complejidad de las aproximaciones y el análisis los resultados obtenidos con distintos conjuntos de datos es el núcleo de la misma. Por este motivo se valorarán todos los criterios enumerados a continuación, pero los tres elementos del primer punto suman el 80% de la valoración total.

- ***Documentación, análisis teórico de la complejidad de los algoritmos y el análisis de los resultados numéricos.***
- Corrección del funcionamiento.
- Corrección del código. Ausencia de *warnings* u otros elementos no deseables como variables no utilizadas, código que no se emplea, etc.
- Documentación de los ficheros fuente. Las explicaciones y razonamientos que se incluyan a modo de comentarios en el código que entrega el alumno. Se solicita que el alumno entregue adicionalmente la documentación generada en formato javadoc, junto con los ficheros de código fuente.