TRABAJO EPD EVALUABLE ALGORÍTMICA I

REALIZADO POR JAIME BAQUERIZO DELGADO (GRUPO 12) Y VICTOR JESÚS REINA LOPEZ (GRUPO 13)

ENTREGADO EL 11 DE NOVIEMBRE DE 2021

Víctor Jesús Reina López
Jaime Baquerizo Delgado
2º GIISI Algorítmica I - 2021/2022

Cuestiones EPD Evaluable - Algorítmica I

Cuestión a). Indique a continuación la fórmula que usará para la implementación de su solución recursiva. Indique su caso base y su caso general. Explique por qué considera que su fórmula es la adecuada para el problema planteado:

Métodos Comparador de vectores f1(v[]):

```
boolean sonIguales(int[] vector1, int[] vector2) {
    //Caso Base
    if (vector1.length == 0) {
        return true;
    }
    //Caso General
    else {
        if (vector1[vector1.length - 1] == vector2[vector2.length - 1]) {
                return sonIguales(Arrays.copyOfRange(vector1, 0, vector1.length - 1),
                Arrays.copyOfRange(vector2, 0, vector2.length - 1));
         } else {
                return false;
        }
```

Víctor Jesús Reina López
Jaime Baquerizo Delgado
2º GIISI Algorítmica I - 2021/2022

```
    }
}
```

Explicación método:

En este método van a entrar como parámetros dos vectores, los cuales vamos a comparar si son iguales. Para ello vamos a ir comparando desde sus ultimas posiciones, de manera que su último elemento estaría en la posición tamaño - 1. Así mismo iremos llamando de manera recursiva el método con subvectores de los ya introducidos primeramente, suprimiendo la última posición ya comparada (Gracias al método Arrays.copyOfRange()).

- Caso general:

En caso de que dichos elementos sean iguales seguirá creando subvectores hasta que lleguemos al caso base, y en caso contrario devolverá un booleano con valor *false* ya que nos basta con un elemento distinto en los vectores para darnos cuenta que son diferentes.

- Caso base:

Por esto, nuestro caso base sería que la dimensión de alguno de los vectores sea 0, ya que estamos creando subvectores idénticos pero sin la última posición ya comparada, así que si estas últimas posiciones son siempre iguales llegará un momento en el que de tanto decrementar el vector su tamaño será 0, y esto significa que los vectores son los mismos y se devolverá un booleano con valor *true*.

Métodos Comparador de matrices:

```
boolean sonMatricesIguales(int[][] matriz1, int[][] matriz2, int fila) {
        //Caso Base
        if (fila - 1 == 0) {
                return sonIguales(matriz1[0], matriz2[0]);
        }
        //Caso General
        else {
        if (sonIguales(matriz1[fila - 1], matriz2[fila - 1])) {
                return sonMatricesIguales(matriz1, matriz2, fila - 1);
        } else {
                return false;
        }

    }
}
```

Explicación método:

En este método vamos a comparar si dos matrices son iguales, es decir, si tienen los mismos elementos en las mismas posiciones. Para ello van a entrar como parámetros las dos matrices y el número de filas totales de las matrices (obviando que ambas

matrices tienen las mismas dimensiones), el cual usaremos como referencia para empezar a comparar fila por fila empezando desde la última hasta la primera. Por lo que iremos llamando de forma recursiva al método introduciendo de nuevo las matrices pero decrementando en 1 el número de filas cada vez que comparemos una.
- Caso general:
Vamos a usar el método explicado anteriormente para comparar los vectores formados por las filas de ambas matrices. En caso de que dichas filas sean idénticas (es decir, el método devuelva *true*) seguiremos invocando al método pero con el número de fila decrementado en 1 para así comparar las siguientes. Y en caso de que el método comparador de las filas devuelva *false* nuestro método comparador de las matrices también devolverá *false* ya que hemos comprobado que nuestras matrices no son iguales.
-Caso base:
Como vamos llamando al método con el número de filas disminuyendo, llegaremos a un caso en el que la filas ya no podremos seguir decrementandolas o nos situaremos en la fila número -1, lo cual esto es imposible. Así que en este caso no nos rentaría seguir restando y devolveríamos el resultado de la comparación de estas últimas filas ya que nos indicaría la solución de la comparación final.

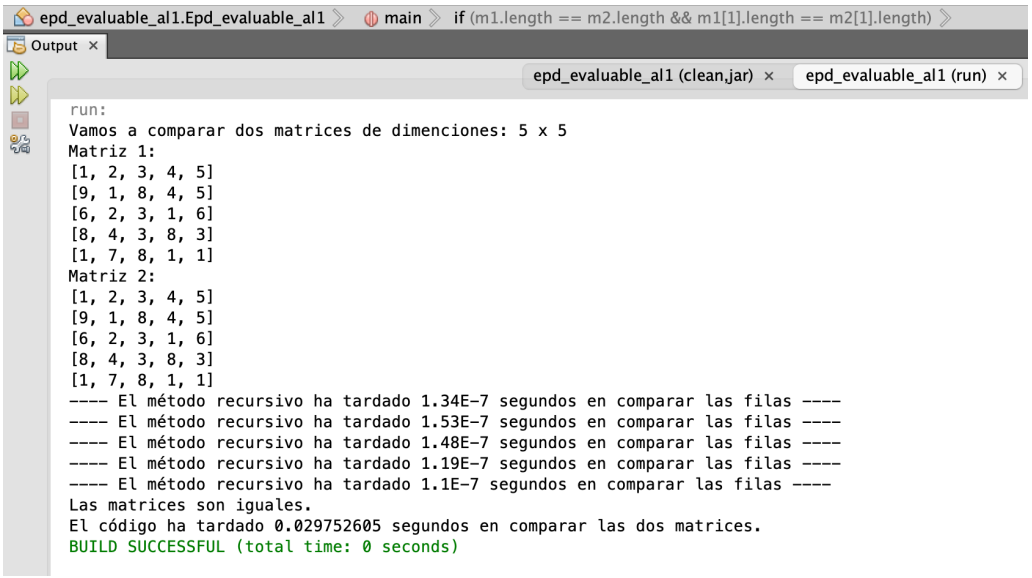
Cuestión b).

- I) Calcule los tiempos de ejecución de su código tal y como se ha hecho en clase. Muestre los tiempos de cada ejecución por consola, cree una gráfica con los tiempos obtenidos y muéstrela a continuación:

Hemos calculado los distintos tiempos de nuestro método desde matrices 2x2 hasta

Víctor Jesús Reina López
Jaime Baquerizo Delgado
2º GIISI Algorítmica I - 2021/2022

unas 100x100, para así apreciar notablemente la incrementación del tiempo de ejecución. Aun así nos damos cuenta que la variación de tiempo entre las dimensiones varía tan solo 1.52 x 10^-2 segundos.
También hemos implementado un cálculo del tiempo de comparación de las filas de nuestras matrices:

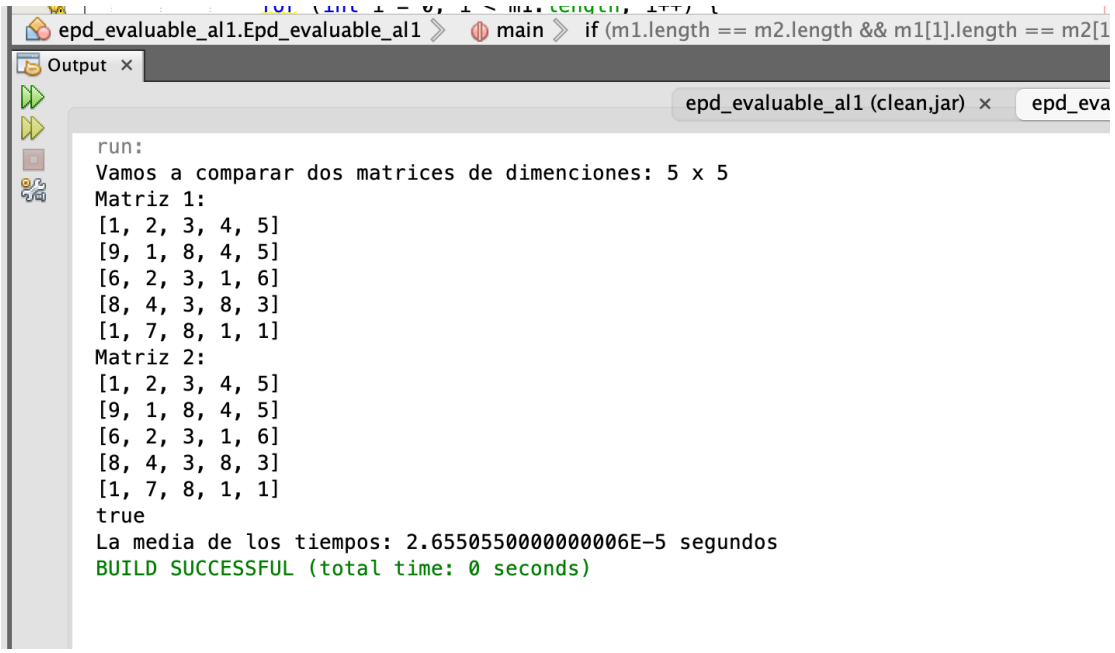


- II) ¿Qué mecanismo debe usar para obtener mayor fiabilidad de los tiempos obtenidos? Coméntelo a continuación (y aplíquelo en su código)

Como nuestro procesador no invierte siempre el mismo periodo de tiempo, no podemos fiarnos de una sola iteración al compilar nuestro código. Para ello vamos a realizar una medida real ejecutando nuestro método *n* repetidas veces y luego hacer una media aritmética de todos estos tiempos obtenidos, para así tener una fiabilidad mayor de nuestro cálculo:

```
12
13      public static void mediaTiempos (int[][] m1, int[][] m2, int filas){
14          double sumatorio = 0;
15          long inicio;
16          long fin;
17          boolean res = true;
18          double tRec;
19          int numIteraciones = 100;
20          for (int i = 0; i < numIteraciones; i++) {
21              inicio = System.nanoTime();
22              res = sonMatricesIguales(m1,m2,filas);
23              fin = System.nanoTime();
24              tRec = ((double) (fin - inicio)) / 1000_000_000;
25              sumatorio += tRec;
26          }
27          double mediaTiempos = sumatorio/numIteraciones;
28
29          System.out.println(res);
30          System.out.println("La media de los tiempos: " + mediaTiempos + " segundos");
31
32      }
33  }
```


Víctor Jesús Reina López
Jaime Baquerizo Delgado
2º GIISI Algorítmica I - 2021/2022

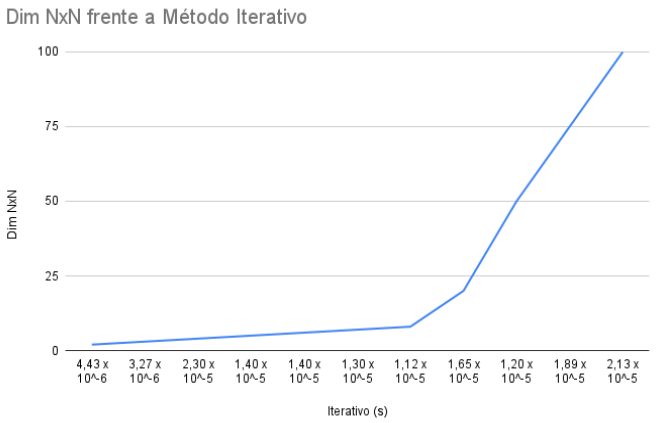


Cuestión c). ¿Cree que una implementación iterativa sería más eficiente en tiempo? ¿Por qué?

Dim NxN	Iterativo (s)	Recursivo (s)
2	4,43 x 10^-6	8,80 x 10^-3
3	3,27 x 10^-6	8,87 x 10^-3
4	2,30 x 10^-5	9,69 x 10^-3
5	1,40 x 10^-5	9,91 x 10^-3
6	1,40 x 10^-5	1,13 x 10^-2
7	1,30 x 10^-5	1,20 x 10^-2
8	1,12 x 10^-5	1,14 x 10^-2
20	1,65 x 10^-5	1,30 x 10^-2
50	1,20 x 10^-5	1,98 x 10^-2

Víctor Jesús Reina López
Jaime Baquerizo Delgado
2º GIISI Algorítmica I - 2021/2022

75	1,89 x 10^-5	2,21 x 10^-2
100	2,13 x 10^-5	2,40 x 10^-2



Implementación iterativa

Víctor Jesús Reina López
Jaime Baquerizo Delgado
2º GIISI Algorítmica I - 2021/2022

```java
    }
    //Método iterativo de comparación de dos Matrices
    public static boolean  sonMatricesIgualesIt(int[][] matriz1, int[][] matriz2, int fila, int columna){

        //Damos por hecho que las matrices son iguales hasta que se demuestre lo contrario en el bucle posterior
        boolean igual = true;
        int i;
        int j;
        i=0;
        //Búsqueda While del elemento distinto de las matrices
        while (i < fila && igual){
            j=0;
            while (j < columna && igual){
                //Nada mas encuentre un elemento diferente pondrá el booleano a false y saldrá de los while
                if(matriz1[i][j] != matriz2[i][j]){
                    igual = false;
                }
                j++;
            }
            i++;
        }

        //Devolvemos el resultado del bucle
        //Si las matricez son iguales nunca llegaría a entrar al "if" y la variable "igual" seguiría a true
         return igual;
    }
```

Los algoritmos iterativos tienden a ser más eficientes que los algoritmos recursivos según un problema en particular, ya que poseen un orden que los hace más eficientes.
Además, se comprueba que un método que se ejecuta *n* cantidad de veces, se logra resolver en menor cantidad tiempo, ya que el compilador al realizar el análisis almacena en memoria un direccionamiento al método ya compilado y analizado sintáctica y semánticamente.
Es preciso emplear algoritmos recursivos cuando no se tengan que realizar múltiples llamadas al mismo método ya que resultan ser ineficientes. Asimismo, es conveniente tomar en cuenta, que se utilicen algoritmos recursivos cuando no se tenga otra opción para solucionar el problema.

Cuestión d). Y en memoria consumida ¿sería más eficiente la implementación iterativa? ¿Por qué?

Cuando programamos recursivamente, todos los resultados parciales se van almacenando en memoria en tiempo de ejecución. Si nuestro árbol recursivo se expande en demasía, es probable que nuestra memoria acabe por agotarse. Por tanto generalmente la implementación recursiva consume bastante más memoria que la iterativa.
Por otro lado, la complejidad para encontrar una solución iterativa a veces es muy elevada. Sobre todo a la hora de implementar fórmulas o soluciones matemáticas (muchas de las cuales se apoyan en la recursividad para definirse), es más fácil alcanzar una solución recursiva que iterativa.