

Práctica 3: Medición experimental de tiempos de ejecución en programas

Objetivos:

- Conocer en la práctica la diferencia en tiempo de ejecución entre dos operaciones (trasposición y multiplicación) de distinta complejidad temporal sobre matrices.

Programa a realizar:

Debéis desarrollar un programa que genere iterativamente matrices de distintos tamaños, y realice dos tipos de operaciones sobre las mismas:

- Trasposición de una matriz.
- Multiplicación de dos matrices.

El programa deberá medir el tiempo necesario para realizar cada tipo de operación, y guardar los resultados en un fichero. Usaremos dicho fichero para realizar con MATLAB, Octave, u otro software de vuestra elección, un análisis básico de los resultados obtenidos mediante su representación gráfica.

Medición de tiempos de ejecución:

Para medir el tiempo de ejecución de un programa disponemos en C de la biblioteca **<time.h>**. Dicha biblioteca dispone de funciones/procedimientos, constantes, tipos de datos y registros que permiten evaluar el número de períodos o “tics” de reloj que consume una secuencia de instrucciones. La función **clock()** permite acceder a dicha información. Su prototipo es el siguiente:

clock_t clock (void)

clock() devuelve el número de “tics” de reloj que han transcurrido desde un determinado evento de referencia (la creación del proceso desde donde se invoca). Mediante **clock()** podemos medir el tiempo transcurrido en una secuencia de instrucciones sin más que invocar dicha función al inicio y al final de dicha secuencia, guardar ambos resultados y restarlos.

El tipo de dato que devuelve **clock()** es el tipo opaco **clock_t**, que en nuestro caso equivale a **long int** (aunque en otros sistemas puede ser **double**).

Para convertir el número de periodos de reloj a segundos de forma fácil y general puede utilizarse la constante

int CLOCKS_PER_SEC;

Práctica 3: Medición experimental de tiempos de ejecución en programas

```

1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int main(int argc, char** argv) {
7     clock_t inicio, fin;
8
9     inicio=clock(); //TOMAMOS LA PRIMERA REFERENCIA
10    /* AQUÍ SE INICIA EL BLOQUE DE INSTRUCCIONES CUYO TIEMPO DE EJECUCIÓN QUEREMOS MEDIR */
11    AQUÍ VA EL CÓDIGO QUE QUEREMOS
12    "CRONOMETRAR"
13    /* AQUÍ FINALIZA EL BLOQUE DE INSTRUCCIONES CUYO TIEMPO DE EJECUCIÓN QUEREMOS MEDIR */
14    fin=clock(); //TOMAMOS LA SEGUNDA REFERENCIA
15
16    //IMPRIMIMOS EL RESULTADO
17    printf("%u\t%lf\n", fin-inicio, (double)(fin-inicio)/CLOCKS_PER_SEC);
18
19    return (EXIT_SUCCESS);
20 }
21

```

Ejemplo: medida experimental de tiempos en la implementación recursiva de la sucesión de Fibonacci.

Implementación poco eficiente:
implementación recursiva.

En este ejemplo debéis ejecutar el proyecto **P3_FibonacciRecursivo**, que incluye la implementación recursiva de la serie de Fibonacci. Dicha serie se define como:

$$f(n) = \begin{cases} 1 & n = 0, 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

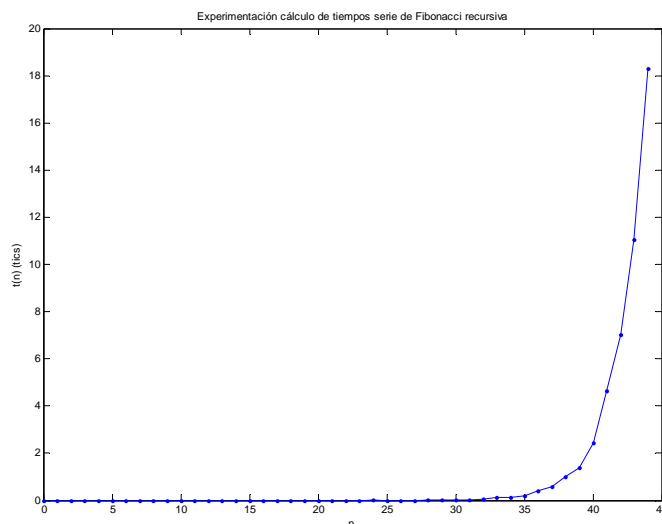
Podéis verificar experimentalmente como progresivamente la obtención de los resultados se ralentiza (debido a que el número de llamadas recursivas, y por lo tanto el tiempo de ejecución, crece exponencialmente)¹. Fijaos, en particular, en el tiempo de obtención de los últimos términos de la sucesión (n=43, 44).

¹ Como curiosidad, comentar que el tiempo de obtención del término n-ésimo t(n) tiende a verificar también la sucesión de Fibonacci: t(n) → t(n-1)+t(n-2) cuando n → ∞.

Práctica 3: Medición experimental de tiempos de ejecución en programas

```
C:\cygwin\bin\sh.exe
n:16 f(n):1597 ini.:31 fin:31 tiempo:0.0000
n:17 f(n):2584 ini.:31 fin:31 tiempo:0.0000
n:18 f(n):4181 ini.:31 fin:31 tiempo:0.0000
n:19 f(n):6765 ini.:31 fin:31 tiempo:0.0000
n:20 f(n):10946 ini.:31 fin:31 tiempo:0.0000
n:21 f(n):17711 ini.:31 fin:31 tiempo:0.0000
n:22 f(n):28657 ini.:31 fin:31 tiempo:0.0000
n:23 f(n):46368 ini.:31 fin:31 tiempo:0.0000
n:24 f(n):75025 ini.:46 fin:31 tiempo:0.0150
n:25 f(n):121393 ini.:46 fin:46 tiempo:0.0000
n:26 f(n):196418 ini.:62 fin:46 tiempo:0.0160
n:27 f(n):317811 ini.:78 fin:62 tiempo:0.0160
n:28 f(n):514229 ini.:93 fin:78 tiempo:0.0150
n:29 f(n):832040 ini.:124 fin:93 tiempo:0.0310
n:30 f(n):1346269 ini.:171 fin:124 tiempo:0.0470
n:31 f(n):2178309 ini.:249 fin:171 tiempo:0.0780
n:32 f(n):3524578 ini.:343 fin:249 tiempo:0.0940
n:33 f(n):5702887 ini.:421 fin:343 tiempo:0.0780
n:34 f(n):9227465 ini.:530 fin:421 tiempo:0.1090
n:35 f(n):14930352 ini.:748 fin:530 tiempo:0.2180
n:36 f(n):24157817 ini.:1092 fin:748 tiempo:0.3440
n:37 f(n):39088169 ini.:1684 fin:1092 tiempo:0.5920
n:38 f(n):63245986 ini.:2605 fin:1684 tiempo:0.9210
n:39 f(n):102334155 ini.:4227 fin:2605 tiempo:1.6220
n:40 f(n):165580141 ini.:6801 fin:4227 tiempo:2.5740
n:41 f(n):267914296 ini.:11091 fin:6801 tiempo:4.2900
n:42 f(n):433494437 ini.:18095 fin:11091 tiempo:7.0040
n:43 f(n):701408733 ini.:28922 fin:18095 tiempo:10.8270
n:44 f(n):1134903170 ini.:46674 fin:28922 tiempo:17.7520
```

El programa genera un fichero de nombre “**tiemposFibonacciRecursivo.txt**” con dos columnas de datos que son respectivamente **n** y **t(n)**. Podéis visualizarlos gráficamente desde Matlab ejecutando el programa **repreTiemposFibo.m**.



Para valorar la importancia de basar la implementación de los programas en algoritmos lo más eficientes posible, podéis experimentar ahora sobre el programa anterior, cambiando la implementación recursiva por esta otra iterativa:

Práctica 3: Medición experimental de tiempos de ejecución en programas

```
unsigned long fibonacciIterativo(unsigned char n){
    unsigned long a=1, b=1, c=-1;
    for(k=2; k<=n; k++){
        c=a+b;
        a=b;
        b=c;
    }
    return b;
}
```

k	a	b	c
2	1	1	-1
2	1	2	2
3	1	2	2
3	2	3	3

En particular, comparad el tiempo invertido en obtener los últimos términos de la sucesión entre las dos implementaciones propuestas: esta nueva implementación iterativa y la anterior implementación recursiva. ¿Cuál es la diferencia para los términos $n > 40$?

Ejercicio 3.1: medida experimental de tiempos de trasposición de matrices.

En el ejercicio vamos a implementar un programa que nos permita medir los tiempos de ejecución para la trasposición de matrices. Utilizaremos el TAD **matrizdinamica** implementado en la práctica 1. Para cada valor de n , el programa medirá el tiempo de ejecución $t(n)$ consumido en realizar la operación de trasposición de una matriz (supondremos tipo **float** para los elementos de la matriz) de tamaño $n \times n/2$. Dado que la matriz pueden tener tamaños muy grandes (> 10.000), es necesario que en algún punto del programa cambies el tipo de alguna variable de **short** (< 32.767) a **long** ($< 2.147.483.647$). Estos cambios dependerán de la implementación interna del TAD matriz.

¿En qué funciones es estrictamente necesario realizar este cambio, asumiendo que el tamaño máximo de la matriz es de 30.000×30.000 ?

Adicionalmente debéis añadir los siguientes procedimientos/funciones al TAD:

- **void inicializar(matrizP *matrix).** Realiza la inicialización de los elementos de la matriz asignándoles valores aleatorios. Para la inicialización se puede utilizar la siguiente línea: `element = 10.0 * (TELEMENTO) rand() / RAND_MAX;`

Práctica 3: Medición experimental de tiempos de ejecución en programas

Cada elemento de la matriz tomará un valor aleatorio entre 0 y 10,0. Posteriormente, se llamará a la función **asignar** para almacenar el valor en la matriz en la posición correspondiente.

- **void trasp(matrizP* result, matrizP m1).** Esta función transpone la matriz *m1*, y almacena el resultado en la matriz *result*. En la función es necesario realizar los siguientes pasos:
 - Comprobar que las dimensiones de la matriz *result* son las adecuadas para almacenar la matriz traspuesta de *m1*.
 - Llamar a la función *asignar* para almacenar cada elemento de la matriz *m1* en la posición correspondiente de la matriz *result*. Para acceder a los valores de la matriz utiliza la función *recuperar*.

Esta operación de trasposición de matrices tiene complejidad cuadrática.

En el programa principal nos encargaremos de realizar la experimentación. Para ello debéis implementar un bucle que vaya desde el tamaño inicial de matriz hasta el tamaño final de matriz, paso a paso (el tamaño del paso lo define el usuario también). En el bucle se realizarán las siguientes tareas:

1. Crear e inicializar la matriz que se va a transponer (de tamaño $n \times n/2$), y crear la matriz donde se almacenará el resultado.
2. Almacenar el tiempo de inicio.
3. Realizar la trasposición de la matriz mediante la función *trasp*.
4. Almacenar el tiempo de fin.
5. Imprimir a pantalla y a fichero el número de filas de la matriz (**n**) y el tiempo **t(n)** que tarda en realizarse la trasposición. El fichero tendrá dos columnas de datos, **n** y **t(n)** (formato idéntico al del ejemplo previo).
6. Liberar las **dos** matrices: la inicial y su traspuesta.

Entre el tiempo de inicio y el tiempo de fin hay una única llamada a una función (*trasp*).

El programa recibirá como argumentos en la línea de entrada los valores de tamaño inicial, tamaño final y paso necesarios para realizar la experimentación. El programa debe poder aceptar cualquier entrada de estos tres valores para poder realizar experimentos con tamaños iniciales, finales y pasos distintos.

Una vez ejecutado el programa, podéis visualizar una gráfica con los resultados **t(n)** frente a **n** utilizando el programa Matlab **generarGraficaTraspuesta.m** que representa los resultados obtenidos ajustándolos a un polinomio de grado 2. Ejecutad el programa para distintos rangos de tamaños de matrices y analizad con cuidado las tendencias obtenidas. El programa Matlab es únicamente para mostrar la gráfica con los resultados que saca a fichero vuestro programa C.

Práctica 3: Medición experimental de tiempos de ejecución en programas

Ejercicio 3.2: medida experimental de tiempos de multiplicación de matrices.

En este ejercicio se trata de repetir la experimentación anterior utilizando la operación de multiplicación de matrices. Para ello, deberéis añadir la siguiente función al TAD:

- **void mult(matrizP* result, matrizP m1, matrizP m2).** Esta función realiza la multiplicación de las matrices $m1$ y $m2$, y almacena el resultado en la matriz *result*. Los tamaños de las matrices $m1$ y $m2$ son, respectivamente $n \times n$ y $n \times n/2$. En la función es necesario realizar los siguientes pasos:
 - Comprobar que las dimensiones de las matrices permiten realizar la operación de multiplicación.
 - Calcular el valor de cada elemento (i, j) de la matriz *result*. Para acceder a los valores de las matrices utiliza la función *recuperar*.
 - Llamar a la función *asignar* para almacenar cada valor (i, j) en la matriz *result*.

Esta operación de multiplicación de matrices tiene complejidad cúbica.

El ejercicio consiste en realizar la misma experimentación que en el ejercicio anterior, para obtener los tiempos de ejecución de la multiplicación de matrices para diferentes tamaños. En la función principal debes añadir la llamada a la función de multiplicación.


Al igual que en el ejercicio anterior, el programa recibirá como argumentos en la línea de entrada los valores de tamaño inicial, tamaño final y paso necesarios para realizar la experimentación. El programa debe poder aceptar cualquier entrada de estos tres valores para poder realizar experimentos con tamaños iniciales, finales y pasos distintos. **Adicionalmente, se pasará por línea de comandos un último argumento que será la operación a realizar: "t" para la trasposición y "m" para la multiplicación.** De esa forma será posible utilizar el mismo programa para resolver los ejercicios 3.1 y 3.2.

Una vez ejecutado el programa, podéis visualizar una gráfica con los resultados $t(n)$ frente a n utilizando el programa Matlab **generarGraficaProducto.m** que representa los resultados obtenidos ajustándolos a un polinomio de grado 3. **Nuevamente, el programa Matlab es únicamente para mostrar la gráfica con los resultados que saca a fichero vuestro programa C.**

Práctica 3: Medición experimental de tiempos de ejecución en programas

Entregables

Se deberá realizar la entrega en el Campus Virtual. Las fechas de entrega se especifican en el Campus Virtual, y las instrucciones son las siguientes:

- En esta práctica hay que realizar dos entregas independientes:
 - El código desarrollado. Al igual que en prácticas anteriores, **el código a entregar estará únicamente compuesto por los archivos fuentes** (main.c, matrizdinamica.c, matrizdinamica.h) **y un Makefile** que compile perfectamente en Linux. Cualquier ejercicio que no compile directamente con el makefile (sin opciones) en Linux será evaluado con la calificación de 0. Este criterio se mantendrá en el resto de prácticas de la asignatura.
 - El informe con la experimentación realizada y el análisis correspondiente. Este informe **debe ser entregado en formato PDF**. La entrega del documento en otros formatos tendrá automáticamente la calificación de 0. El informe será un breve documento (2 páginas en total) introduciendo el problema de experimentación, comentando los resultados, mostrando las gráficas y contestando a preguntas como: ¿qué tamaños de matrices son gestionables en tiempo “razonable” por cada algoritmo?
- El fichero comprimido  conteniendo el código y el PDF tendrán el nombre apellido1_apellido2 y la extensión .ZIP y .pdf respectivamente.

Algunos errores típicos detectados en años anteriores y que debéis evitar a la hora de redactar el informe (**serán tenidos en cuenta en la revisión del informe**):

- Gráficas caóticas por no probar rangos de tamaños suficientemente grandes. No se aprecia crecimiento cuadrático o cúbico.
- Errores de edición del informe (p. ej. falta de justificación derecha de los párrafos, falta de nombre de autor en el informe, errores ortográficos, falta de título en el documento, falta de introducción, edición no profesional –tipos de fuente heterogéneos, etc-).
- alguna de las preguntas planteadas en el guión no es contestada en el informe o, si se hace, es de forma genérica, sin proporcionar datos experimentales.
- Argumentar en el informe que cuadrático=lineal, o que cuadrático=cúbico, es decir, tener confusión respecto a las clases de complejidad.
- La cronometración no cuenta sólo el coste de ordenar sino también alguna operación adicional (creación, relleno de aleatorios o liberación de las matrices, por ejemplo).

Práctica 3: Medición experimental de tiempos de ejecución en programas

Evaluación

La evaluación de la práctica será de 13 puntos, divididos en dos partes: 7 puntos del test de autoevaluación, y 6 puntos del informe entregado. A su vez, el informe entregado se calificará como sigue:

- 4 puntos se calcularán a partir de la evaluación por pares que realizarán vuestros compañeros de vuestro informe, siguiendo los criterios que se enumeran a continuación.
- Los 2 puntos restantes se asignarán en función de lo correcta que sea la evaluación que hagáis de los trabajos de vuestros compañeros.

Los criterios de evaluación de la revisión por pares son los siguientes:

1. La presentación del informe es adecuada: los párrafos tienen justificación derecha, se incluye el nombre del autor en el informe, no hay errores ortográficos, se incluye el título en el documento, la edición es profesional –tipo de fuente homogéneo, etc.
2. El informe está completo: contiene introducción, las dos gráficas y comentarios sobre ambas gráficas.
3. La gráfica de la trasposición es correcta: se prueba con rangos de tamaños suficientemente grandes, y se aprecia crecimiento cuadrático.
4. Los comentarios sobre la gráfica de la trasposición son correctos: se indica el rango probado, se argumenta por qué se ha seleccionado dicho rango y se justifica por qué el crecimiento es cuadrático.
5. La gráfica de la multiplicación es correcta: se prueba con rangos de tamaños suficientemente grandes, y se aprecia crecimiento cúbico.
6. Los comentarios sobre la gráfica de la multiplicación son correctos: se indica el rango probado, se argumenta por qué se ha seleccionado dicho rango y se justifica por qué el crecimiento es cúbico.