Práctica 4. Divide y Vencerás

Objetivos

Practicar la estrategia algorítmica Divide y Vencerás para un problema real.

Programa a realizar

Debéis desarrollar un programa que genere aleatoriamente 2 matrices cuadradas n x n (siendo n una potencia de 2) y las multiplique siguiendo una conocida estrategia de Divide y Vencerás.

Pasos a seguir

Se partirá del TAD **matrizdinamica** desarrollado en la práctica 1 y modificado en la práctica 3.

El programa principal recibirá como argumento desde línea de comandos el tamaño de las matrices *n* (debe ser potencia de dos), y realizará las siguientes operaciones:

1. Creará 4 matrices: las dos que se van a multiplicar y las dos matrices donde se almacenará el resultado de la multiplicación por el método convencional y por el método de Divide y Vencerás. Además, inicializará las matrices a multiplicar mediante la función del TAD **inicializar** (práctica 3), y las imprimirá por pantalla mediante la función **imprimir** del programa principal.
2. Multiplicará las dos matrices mediante la estrategia de Divide y Vencerás. Para ello se realizará una llamada a una nueva función del TAD:

   void mult_divide_venceras(matrizD* result, matrizD m1, matrizD m2)

   A continuación se imprimirá por pantalla el resultado.
3. Multiplicará las dos matrices de forma convencional utilizando la función del TAD **mult** desarrollada en la práctica 3. A continuación se imprimirá por pantalla el resultado.
4. Comprobará que los resultados de la multiplicación mediante ambos métodos es igual utilizando una nueva función del TAD:

   int matricesIguales(matrizD m1, matrizD m2)

   A continuación imprimirá por pantalla desde el programa principal un mensaje indicando si el resultado de la multiplicación es o no el mismo.

Práctica 4. Divide y Vencerás

Nuevas funciones a desarrollar en el TAD

En el TAD es necesario desarrollar tres nuevas funciones:

- void mult_divide_venceras(matrizD* result, matrizD A, matrizD B)

  Realizará la multiplicación de las matrices *A* y *B* mediante la estrategia de Divide y Vencerás, y almacenará el resultado en la matriz *result*. Esta función se explica en detalle en la siguiente sección.

  La multiplicación de matrices mediante Divide y Vencerás requiere que el tamaño de las matrices sea potencia de 2. La comprobación más sencilla se puede realizar a nivel de bit: un número que es potencia de dos tendrá en binario un 1 y el resto de cifras serán 0. Por ejemplo, el 8 será: 1000. Si a ese número le restamos 1, el número resultante en binario implica cambiar todos los 0 por 1 y los 1 por 0. Así, 8-1 será: 0111. Si aplicamos el operador & (*and* a nivel de bit), el resultado será 1000&0111=0000. Por tanto, para comprobar si el tamaño de la matriz cuadrada A es potencia de 2 se podría utilizar el siguiente código: if((nfilas(A) & (nfilas(A)-1)) == 0)

- void inicializarSubMatriz(matrizD *submatriz, matrizD original, int cuadrante)

  Tal y como se especifica en la siguiente sección, al multiplicar mediante Divide y Vencerás es necesario crear una serie de submatrices. La función **inicializarSubMatriz** recibe como argumentos una submatriz ya creada que necesita ser inicializada, la matriz original y un código para indicar qué cuadrante de la matriz original va a ser copiado en la submatriz. El cuadrante 1 es el superior izquierdo, el 2 el superior derecho, el 3 el inferior izquierdo y el 4 el inferior derecho. La inicialización de la submatriz se realiza copiando las correspondientes posiciones mediante las funciones necesarias del TAD.

- void suma(matrizD* result, matrizD A, matrizD B)

  Realizará la suma de las matrices *A* y *B*, y almacenará el resultado en la matriz *result*.

- int matricesIguales(matrizD m1, matrizD m2)

  Esta función determina si dos matrices son iguales. Dado que TELEMENTO es de tipo float, es necesario definir una precisión para determinar si dos números son iguales. Para ello, calcularemos su diferencia en valor absoluto y comprobaremos si es superior a la precisión requerida mediante la siguiente línea de código:

  fabs(valor1 – valor2) > precision

  Donde *valor1* y *valor2* son dos variables de tipo TELEMENTO (float en nuestro caso) y *precision* es una constante que tomará el valor 0,01.

  Para el programa completo, ¿qué ocurre si *precision* toma el valor de 0,0001 y el tamaño de matriz es grande? ¿Por qué influye el tamaño de la matriz en el resultado? ¿Qué

modificación sería necesaria para que el programa funcione correctamente para un valor de *precision* de 0,0001?

Práctica 4. Divide y Vencerás

Multiplicación de matrices mediante la estrategia de Divide y Vencerás

Como sabéis, la técnica de "Divide y Vencerás" realiza los siguientes pasos:

1.- Descomponer el problema a resolver en subproblemas más pequeños.

2.- Resolver independientemente cada subproblema.

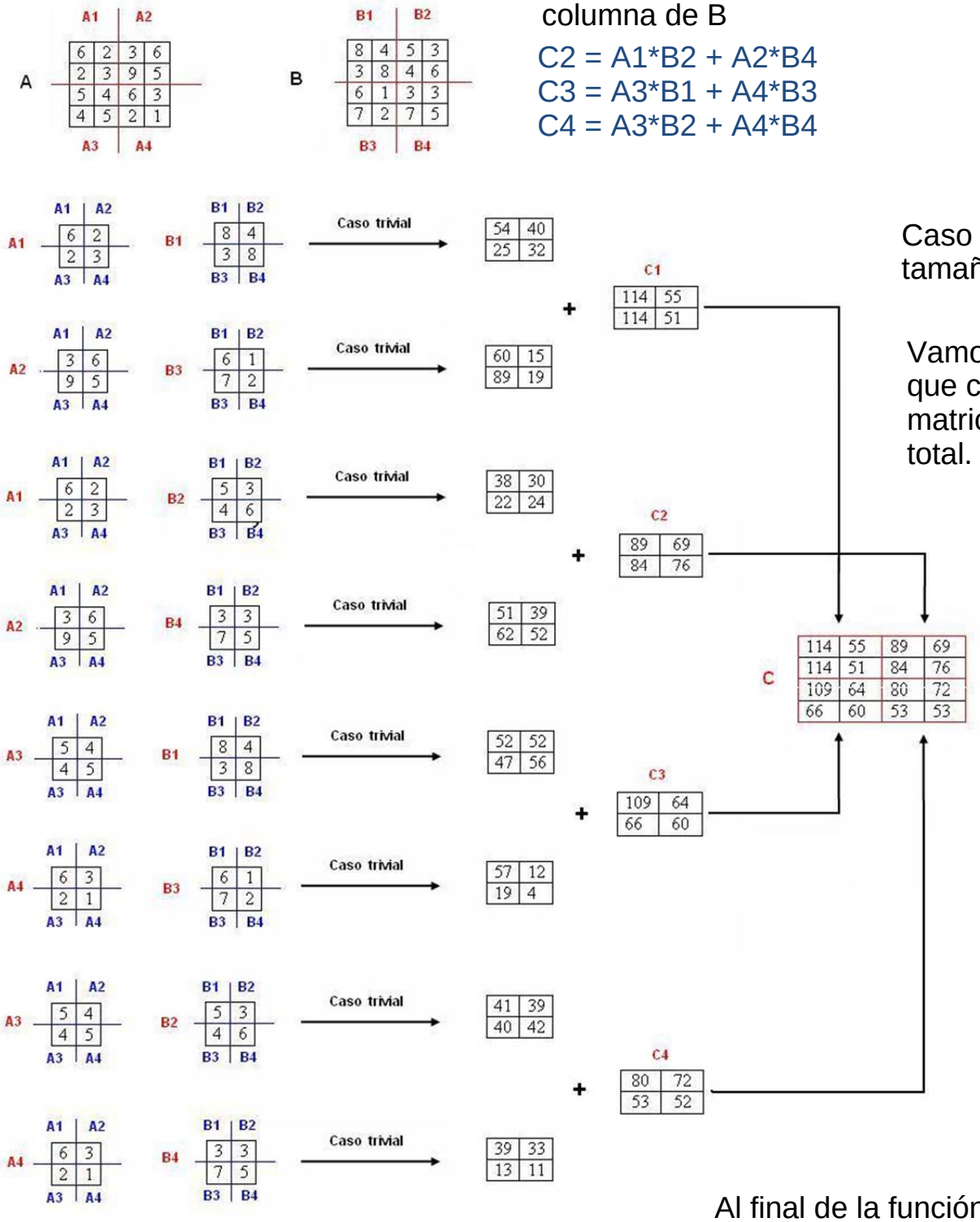
3.- Combinar los resultados obtenidos para construir el problema original.

En el caso de la multiplicación de matrices, el problema inicial tendrá un tamaño de n, y se dividirá en otros 4 de tamaño n/2, cada uno de éstos en otros 4 de tamaño n/4 y así sucesivamente. Gráficamente:

Práctica 4. Divide y Vencerás

C1 = A1*B1 + A2*B3
Multiplicación 1ª fila de A por 1ª columna de B

C2 = A1*B2 + A2*B4
C3 = A3*B1 + A4*B3
C4 = A3*B2 + A4*B4



Caso trivial: tamaño 2x2

Vamos a tener que crear 13 matrices en total.

Al final de la función divide y vencerás hay que liberar las 13 matrices.

Práctica 4. Divide y Vencerás

La funcion de multiplicación mediante Divide y Vencerás tendrá el siguiente esquema:

```
void mult_divide_venceras(matrizD* result, matrizD A, matrizD B)
{
// se entiende que A, B y result son tam x tam
// el tamaño lo podéis obtener a partir del TAD matrizdinamica

// pasos a seguir....

/* si tam<=2 no se va a subdividir más el problema, tratándose como un caso base. Realizaremos directamente la multiplicación */

if (tam<=2)
        {
        // Se realizará la multiplicación llamando a la función mult desarrollada en la práctica previa.
        .....

        }
else
        {
        // divide y vencerás

        // creamos 4 matrices de tamaño n/2 para representar las 4 submatrices de A
        ....
        // creamos 4 matrices de tamaño n/2 para representar las 4 submatrices de B
        ....

        //lo anterior implica crear las matrices y copiar la parte correspondiente de la matriz
        /*Esto hay que hacerlo mediante una  función auxiliar dentro del TAD */
        /*Esta función recibe la submatriz ya creada, la matriz original y un código para indicar qué "cuadrante" quieres copiar, y se ocupa de llamar a las operaciones del TAD matrizdinamica para copiar los valores */


        // Invocaciones recursivas para realizar productos individuales entre submatrices. Por cada invocación hay que crear previamente la matriz donde se almacenará el resultado.
        ....

        /* suma de matrices para conseguir resultado final
        Habrá que crear una nueva función que realice la suma de dos matrices*/

        ....
```

```
        // copia de los cuatro cuadrantes del resultado en la matriz resultado

        // Destrucción de todas las matrices intermedias que fue necesario crear
        .....
        }


}
```

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:

- 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.
- Se entregará un único fichero comprimido conteniendo el código. El archivo tendrá el nombre apellido1_apellido2 y la extensión .zip.