UNIVERSIDAD DE GRANADA

Práctica 2
Algoritmos Divide y Vencerás

Joaquín Sergio García Ibáñez
Juan Navarro Maldonado

ÍNDICE:

Ejercicio 1

La unidad aritmético-lógica (ALU) de un microcontrolador de consumo ultrabajo no dispone de la operación de multiplicación. Sin embargo, necesitamos multiplicar un número natural i por otro número natural j, que notamos como i*j. Sabemos, por la definición matemática de la operación de multiplicación en los números naturales, que:

$$i*j=\begin{cases} i & j=1 \\ i*(j-1)+i & j>1 \end{cases} \forall\, i,j \in N$$

Asumiendo como método básico la implementación directa de la fórmula anterior: ¿Es posible crear un algoritmo Divide y Vencerás más eficiente que la implementación de este algoritmo básico? Se asume que el número de bits de la ALU es desconocido pero fijo (por tanto, no se trata del problema de multiplicación de enteros largos).

Sea el problema P a resolver, de tamaño n, que consiste en multiplicar dos números mediante el uso de la fórmula anterior.

Para realizar este ejercicio, implementaremos dis arrays v1[0..n] y v2[0..n] de números aleatorios de 0 a 10 del tamaño que se quiera, para posteriormente multiplicarlo componente a componente.

Algoritmo Básico

En este apartado, diseñaremos un algoritmo básico para la resolución de este problema:

Algoritmo Básico (i , j)

SI (j == 1)
ENTONCES:
    resultado = i
EN OTRO CASO SI (j > 1)
    resultado = i *(j-1)+i;

DEVOLVER resultado

Diseño de componentes

División del problema en subproblemas

Divido el problema P, de tamaño n, en K=2 subproblemas que son:

P1: Es el subvector v1[1 .. n/2] y v2[1 .. n/2], de tamaño n1=n/2

P2: Es el subvector v[n/2+1 .. n] y v2[n/2+1 .. n], de tamaño n2 = n - n1

Los subproblemas P1 y P2 son de la misma naturaleza que P (multiplicar las componentes de dos vectores), de aproximadamente el mismo tamaño n1 = n2 = n/2, y son independientes.

Combinación de subproblemas

Sea S la solución al problema original P, S1 la solución de P1 y S2 la solución de P2. La solución S se calcula como la fusión de S1 y S2 de la siguiente forma:

S = combina(S1,S2)

La solución S se calcula como la unión de los dos vectores en uno solo, de esta forma, tendríamos un nuevo vector en el cual se almacenan la solución de la multiplicación de cada componente de los dos vectores. Por ejemplo para la componente S[0] = v1[0]*v2[0]

Caso Base

El caso base se da cuando n = 1 (ambos vectores de una sola componente). La solucion S se calcula como:

S = v1*v2

Diseño del Algoritmo

Algoritmo S = multiplicaDyV(Problema P:v1[1..n],v2[1...n)

SI n = 1:
    S = AlgoritmoBasico(v1,v2)
SI NO:
    Dividir el problema en 2 subproblemas
    P1: v1[1..n] v2[1..n]
    P2: v1[n/2+1 .. n] v2[n/2+1 .. n]
    Resolver cada subproblema
    S1 = CuadradoPerfectoDyV(P1)
    S2 = CuadradoPerfectoDyV(P2)
    S = combina(S1,S2)

DEVOLVER S

Algoritmo S = combina(S1, S2)

```
MIENTRAS i < tam
    SI i < tam/2
        S[i] = S1[i]
    EN OTRO CASO
        S[i]=S2[i]
    i = i+1
```

Análisis de eficiencia

El tamaño del problema depende de un parámetro tam, que es la longitud del vector de componentes y a la vez es el número que queremos multiplicar.
Podemos observar que se trata de una sentencia condicional if - else, el if tiene eficiencia O(1) y el else viene compuesto por 4 sentencias de asignación de eficiencia O(1) y 2 sentencias recursivas las cuales tienen tamaño n/2 y para averiguar las eficiencias de cada llamada recursiva debemos de sacar las ecuaciones y como son 2 llamadas recursivas seria:

$$T(n) = 2T(n/2)$$

Resolvemos la ecuación por cambio de variable $n = 2^m$

$$T(2^m) = 2T(2^m)$$

$$T(2^m) - 2T(2^m) = 0$$

$$x^{2^m} - 2x^{2^{m-1}} = 0$$

$$x^{2^{m-1}}(x - 2) = 0$$

$$R_1 = 2 \qquad M_1 = 1$$

$$t_{2^m} = C_{10}2^m * m$$

Deshacemos el cambio de variable

$$t_n = C_{10}n * log(n)$$

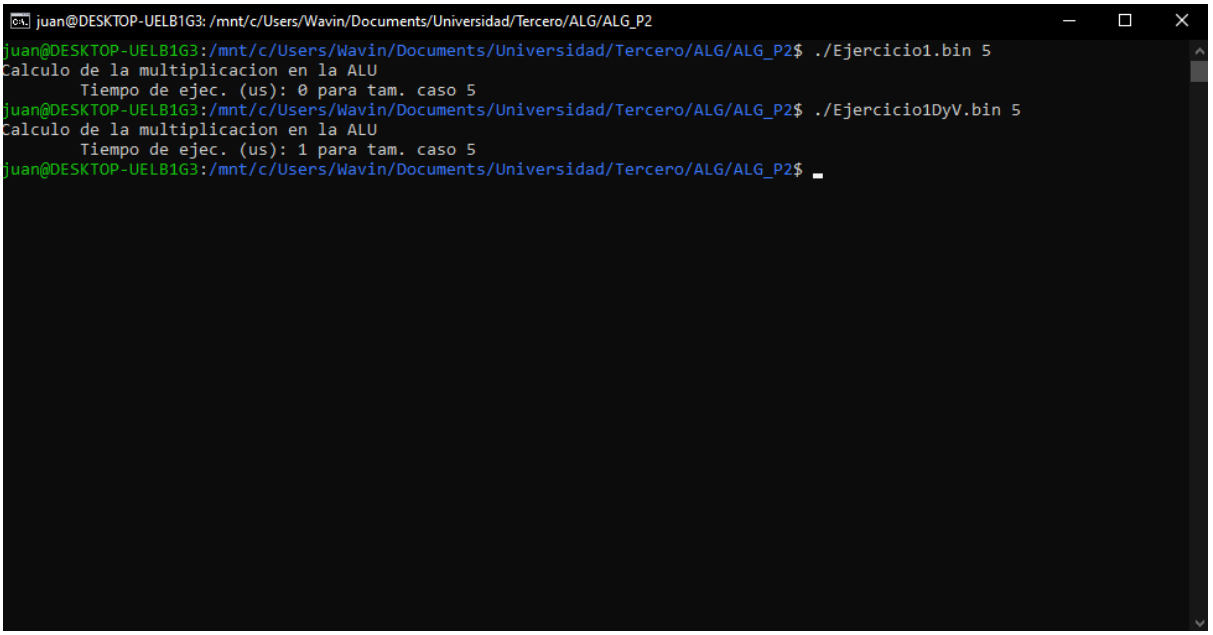
Con lo cual nuestra eficiencia para las llamadas recursivas sería $O(n * log(n))$ y finalmente la eficiencia de todo el algoritmo sería $O(n * log(n))$.

El método combina es de eficiencia O(1).

La eficiencia del algoritmo del caso base tenemos que el tamaño de caso de este algoritmo viene dado por tam el cual va a usar el bucle while tam veces ya que va a ejecutar mientras i sea menos que tam y lo de dentro tiene eficiencia O(1) con lo cual la eficiencia de nuestro algoritmo base será de O(n)

Ejemplos de uso de ambos algoritmo

En este ejercicio implementamos un algoritmo DyV pero no conseguimos una mejora significante para que resultara factible usar un algoritmo DyV en vez del de fuerza bruta ya que al ser tan simple resulta complicado implementar un algoritmo DyV.



Ejercicio 2

Un número natural n se dice que es cuadrado perfecto si se corresponde con el cuadrado de otro número natural. Por ejemplo, 4 es perfecto dado que 4=22 . También son cuadrados perfectos 25=52, o 100=102.

¿Qué método básico/por fuerza bruta podemos diseñar para calcular si un número es cuadrado perfecto o no? ¿Es posible diseñar un algoritmo Divide y Vencerás para igualar (o mejorar) la eficiencia de este método básico?

Sea P, el problema a resolver, de tamaño n, que consiste en comprobar un array ordenado desde 0 a n para saber si es un cuadrado perfecto.
Determinaremos que un número sea cuadrado perfecto siempre y cuando encontremos otro número que multiplicado por sí mismo nos de el número que buscamos.

Algoritmo Básico

En este apartado, diseñaremos un algoritmo básico para la resolución de este problema:

```
Algoritmo Básico(numero)
        cuadrado = incorrecto
        MIENTRAS i <= numero Ó cuadrado = correcto
                SI (i*i == numero)
                        cuadrado = correcto
        DEVOLVER cuadrado
```

Diseño de componentes

División del problema en subproblemas

Dividimos el problema P, de tamaño n, en K=2 subproblemas que son:

P1: Es el subvector v[1..n/2], de tamaño n1 = n/2
P2: Es el subvector v[n/2+1 .. n] de tamaño n2 = n-n1

Los subproblemas P1 y P2 son de la misma naturaleza que P, de aproximadamente el mismo tamaño n1 = n2 = n2/2, y son independientes, ya que el resultado puede encontrarse en cualquiera de los dos subproblemas.

Combinación de subproblemas

S es la solución al problema original P, S1 la solución de P1 y S2 la solución a P2, por lo tanto S sería la fusión de S1 con S2

S=combinacion(S1,S2)

La solución de S se calcula de forma que una vez hallado en S1 o S2 la solución a nuestro cuadrado perfecto se devolverá esa solución como si fuese S.

Caso Base

Se da cuando n = 1 (vector de una componente). La solución la calculo como:
S = v[k] (número entero que sería la raíz del cuadrado).

Diseño del Algoritmo

Algoritmo S = CuadradoPerfectoDyV(Problema P: v[0 .. n])

```
SI n = 1:
        SI v[0] * v[0] == n
                S = v[0]
        EN OTRO CASO
                S = -1
SI NO:
        Dividir el problema en 2 subproblemas
        P1: v[1 .. n/2]
        P2: v[n/2 .. n]
        Resolver cada subproblema
        S1 = CuadradoPerfectoDyV(P1)
        S2 = CuadradoPerfectoDyV(P2)
        S = combina(S1,S2)
DEVOLVER S
```

Algoritmo S = combina(S1, S2)

```
SI S1 >= 0
        DEVOLVER S1
EN OTRO CASO SI S2>=0
        DEVOLVER S2
EN OTRO CASO
        DEVOLVER no hay solución
```

Análisis de eficiencia

El tamaño del problema depende de un parámetro n, que es la longitud del vector de componentes y a la vez es el número que queremos comprobar si es un cuadrado perfecto. Podemos observar que se trata de una sentencia condicional if - else, el if tiene eficiencia O(1) y el else viene compuesto por 4 sentencias de asignación de eficiencia O(1) y 2 sentencias recursivas las cuales tienen tamaño n/2 y para averiguar las eficiencias de cada llamada recursiva debemos de sacar las ecuaciones y como son 2 llamadas recursivas seria:

$$T(n) = 2T(n/2)$$

Resolvemos la ecuación por cambio de variable $n = 2^m$

$$T(2^m) = 2T(2^m)$$
$$T(2^m) - 2T(2^m) = 0$$
$$x^{2^m} - 2x^{2^{m-1}} = 0$$
$$x^{2^{m-1}}(x - 2) = 0$$
$$R_1 = 2 \qquad M_1 = 1$$
$$t_{2^m} = C_{10}2^m * m$$

Deshacemos el cambio de variable

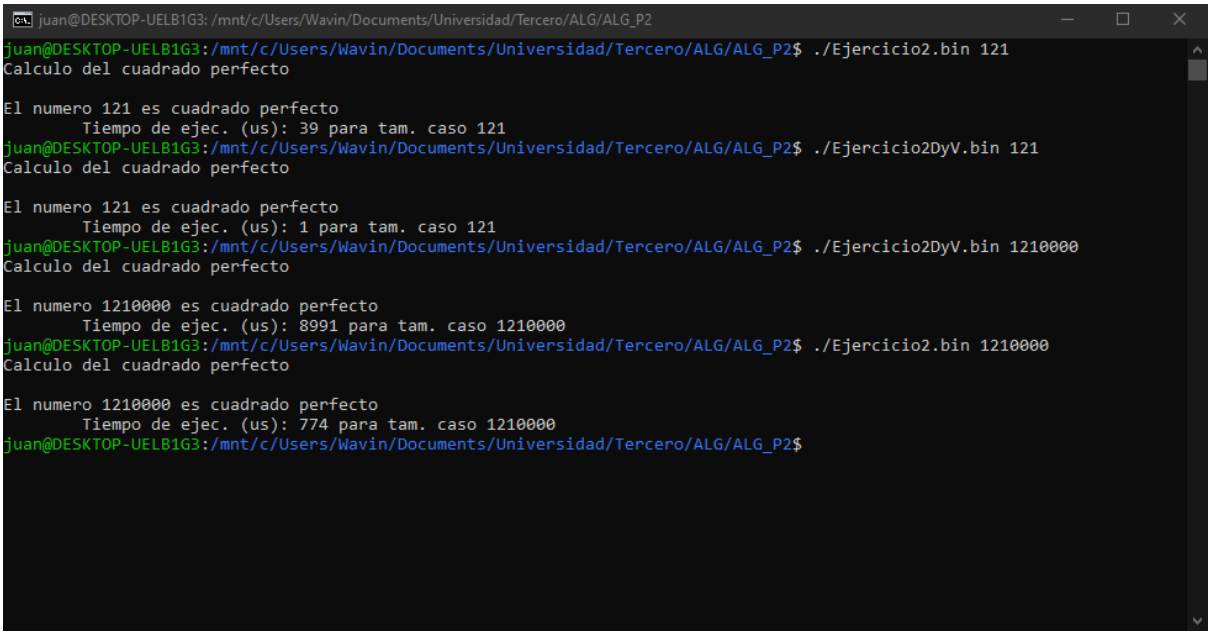
$$t_n = C_{10}n * log(n)$$

Con lo cual nuestra eficiencia para las llamadas recursivas sería $O(n * log(n))$ y finalmente la eficiencia de todo el algoritmo sería $O(n * log(n))$.
El método combina es de eficiencia O(1).

La eficiencia del algoritmo del caso base tenemos que el tamaño de caso de este algoritmo viene dado por el número el cual va a usar el bucle numero veces y lo de dentro tiene eficiencia O(1) con lo cual la eficiencia de nuestro algoritmo base será de O(n);

Ejemplos de uso de ambos algoritmos

En este ejercicio hemos podido encontrar un algoritmo DyV el cual es más eficiente en la práctica que el algoritmo de fuerza bruta podemos observar que para valores pequeños es mucho más rápido el algoritmo DyV que el algoritmo de fuerza bruta.



Ejercicio 3

Dado un número natural n, deseamos saber si existe otro número natural y de modo que
n=y*(y+1)*(y+2). Por ejemplo, para n=60, existe y=3 de modo que
y*(y+1)*(y+2)=3*4*5=60.

¿Qué método básico/por fuerza bruta podemos diseñar para calcular, dado un valor de n de entrada, si existe el número y que hace cumplir la igualdad y cuál es su valor? ¿Es posible diseñar un algoritmo Divide y Vencerás para igualar (o mejorar) la eficiencia de este método básico?

Sea P el problema a resolver, de tamaño n, que consiste en comprobar un array ordenado ascendentemente desde 0 hasta n.
Para determinar que esta igualdad se cumple, se comprobará cada componente del array hasta que se determine si se puede o no cumplir dicha condición.

Algoritmo Básico

El algoritmo básico capaz de resolver nuestro problema sería el siguiente:

Algoritmo Básico(numero, i)
    Mientras que n != resultado y i < n **hacer:**
        ++i
        resultado = i*(i+1)*(i+2)
    Devolver resultado

Diseño de componentes

División del problema en subproblemas

Dividimos el problema P, de tamaño n, en K=2 subproblemas que son:
    P1: Es el subvector v[1..n/2], de tamaño n1 = n/2
    P2: Es el subvector v[n/2+1 .. n] de tamaño n2 = n-n1

Los subproblemas P1 y P2 son de la misma naturaleza que P, de aproximadamente el mismo tamaño n1 = n2 = n2/2, y son independientes, ya que el resultado puede encontrarse en cualquiera de los dos subproblemas.

Combinación de subproblemas

S es la solución al problema original P, S1 la solución de P1 y S2 la solución a P2, por lo tanto S sería la fusión de S1 con S2

S=combinacion(S1,S2)

La solución de S se calcula de forma que una vez hallado en S1 o S2 la solución a nuestro cuadrado perfecto se devolverá esa solución como si fuese S.

Caso Base

El caso base se daría cuando S = v[0]*(v[0]+1)*(v[0]+2), siendo S = n

Diseño del Algoritmo

Algoritmo S = ResuelveDyV(Problema P: v[0 .. n])

SI v[0] = 1:
    S = v[0]*(v[0]+1)*(v[0]+2)
    SI S = n
        S = v[0]
    EN OTRO CASO
        S= -1
SI NO:
    Dividir el problema en 2 subproblemas
    P1: v[1 .. n/2]
    P2: v[n/2 .. n]
    Resolver cada subproblema
    S1 = ResuelveDyV(P1)
    S2 = ResuelveDyV(P2)
    S = combina(S1,S2)
DEVOLVER S

Algoritmo S = combina(S1, S2)
SI S1 >= 0
    DEVOLVER S1
EN OTRO CASO SI S2>=0
    DEVOLVER S2
EN OTRO CASO
    DEVOLVER no hay solución

Análisis de eficiencia

El tamaño del problema depende de un parámetro n, que es la longitud del vector de componentes.
Podemos observar que se trata de una sentencia condicional if - else, el if tiene eficiencia O(1) y dentro de ese if hay un if-else de eficiencia O(1) y el else viene compuesto por 4 sentencias de asignación de eficiencia O(1) y 2 sentencias recursivas las cuales tienen tamaño n/2 y para averiguar las eficiencias de cada llamada recursiva debemos de sacar las ecuaciones y como son 2 llamadas recursivas seria:

$$T(n) = 2T(n/2)$$

Resolvemos la ecuación por cambio de variable $n = 2^m$

$$T(2^m) = 2T(2^m)$$
$$T(2^m) - 2T(2^m) = 0$$
$$x^{2^m} - 2x^{2^{m-1}} = 0$$
$$x^{2^{m-1}}(x - 2) = 0$$
$$R_1 = 2 \qquad M_1 = 1$$
$$t_{2^m} = C_{10}2^m * m$$

Deshacemos el cambio de variable

$$t_n = C_{10}n * log(n)$$

Con lo cual nuestra eficiencia para las llamadas recursivas sería $O(n * log(n))$ y finalmente la eficiencia de todo el algoritmo sería $O(n * log(n))$.

Con lo cual nuestra eficiencia para las llamadas recursivas sería $O(n * log(n))$ y finalmente la eficiencia de todo el algoritmo sería $O(n * log(n))$.
El método combina es de eficiencia O(1).

La eficiencia del algoritmo del caso base tenemos que el tamaño de caso de este algoritmo viene dado por el número el cual va a usar el bucle numero veces y lo de dentro tiene eficiencia O(1) con lo cual la eficiencia de nuestro algoritmo base será de O(n);

Ejemplos de uso de ambos algoritmos

En este ejercicio encontramos también un algoritmo DyV más eficiente que el algoritmo de fuerza bruta para tamaños de casos más pequeños como en el ejercicio anterior.