



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Sesión 4: Divide y vencerás

Integrantes:

Escutia López Arturo

López Santiago Daniel

26/Sept/2016

Teoría

La estrategia de divide and conquer es un paradigma para diseñar algoritmos que se basan en las llamadas a funciones recursivas que derivan a mas ramas que son subproblemas del problema original. Como su nombre lo indica consiste dividir el problema en 2 o más subproblemas del mismo tipo hasta que éstos sean tan sencillos que puedan ser resueltos de manera directa, una vez obtenidos los resultados de los subproblemas se combinan para dar la solución al problema original.

Relación de recurrencia

Los tiempos de complejidad de una función recursiva pueden ser más difíciles de resolver que los algoritmos estándar ya que se necesita saber la complejidad de los subproblemas que están decreciendo de tamaño.

Las relaciones de recurrencia nos dan una herramienta eficaz para calcular los tiempos exactos de complejidad incluyendo constantes. Una relación de recurrencia es una función $t(n)$ definida en términos previos de los valores de n .

Ejemplo

Factorial(n)

If $n=0$ return 1

Else return factorial($n-1$) n

La relación de recurrencia seria $C(n) = C(n-1) + 3$

La multiplicación de karatsuba consiste en realizar multiplicaciones de dígitos muy grandes de una manera más eficiente a la conocida por todos. Primeramente se dividen las longitudes de los dos valores X y Y recursivamente, de esta forma usando tres multiplicaciones de números más pequeños, cada uno la mitad de los dígitos de X y Y , junto con algunas sumas y corrimientos, se obtiene el resultado de la multiplicación original. Aquí claramente se realiza la aplicación de divide y vencerás.

Ejemplo: Supongamos que $X=10111$ y $Y=110$

Paso 1.- Dividir por la mitad la longitud mayor de entre los dos dígitos, en caso de ser un número impar hacer techo(ceiling) al momento de dividir.

$$a=010 \ b=111 \ c=000 \ d=110$$

Paso 2.- Se calcula $a*c$ recursivamente dividiendo de nuevo a y c como un nuevo X y Y hasta que sean multiplicaciones de números de un solo dígito

$$P1=a*c=(010)(000)=2*0=0$$

Paso 3.- Se calcula $b*d$ Se calcula $b*d$ recursivamente dividiendo de nuevo b y d como un nuevo X y Y hasta que sean multiplicaciones de números de un solo dígito

$$P2=b*d=(111)(110)=7*6=42$$

Paso 4.- Se calcula $(a+b)(c+d)$ recursivamente dividiendo de nuevo $a+b$ y $c+d$ como un nuevo X y Y hasta que sean multiplicaciones de números de un solo dígito.

$$P3=(a+b)(c+d)=(010+111)(000+110)=(7+2)(6)=54$$

Paso 5.- Obtener el valor de la multiplicación mediante la siguiente formula y usando el truco de Gauss.

$$XY = p1*2^{n+2n/2} * [p3 - p2 - p1] + p2$$

$$23*6=(0)+(2^3)(54-42-0)+42=138$$

De esta forma se realiza la multiplicación de Karatsuba mediante manejo de bits.

Anatolii Karatsuba aportaciones

- Algoritmo de multiplicación rápida
- Artículos importantes en teoría de autómatas finitos
- En sumas trigonométricas e integrales
- La función ζ de Riemann
- Caracteres de Dirichlet

Carl Friedrich Gauss fue un matemático alemán que contribuyó significativamente en muchos campos, incluida la teoría de números, el análisis matemático, la geometría diferencial, la estadística, el álgebra, la geodesia, el magnetismo y la óptica, así como algunas publicaciones como lo fueron:

- Disertación sobre el teorema fundamental del álgebra
- Disquisitiones Arithmeticae
- Theoria Motus Corporum Coelestium in sectionibus conicis solem ambientium
- Theoria combinationis observationum erroribus minimis obnoxiae.

Método de Substitución: Consta de 2 pasos Deducir la forma de la solución y Utilizar inducción matemática para encontrar las constantes y mostrar que la solución funciona

Método de Árbol de Recursión:

- Dibujar un árbol de recursión
- Cada nodo representa el costo de un subproblema en el conjunto de llamadas a funciones recursivas
- Sumamos costos por nivel y determinamos el costo total de todos los niveles de recursión

Teorema Maestro: En todos los casos compara $f(n)$ con $n^{\log_b(a)}$

Caso 1: $n^{\log_b a}$ es la mayor, la solución es $T(n) = \Theta(n^{\log_b a})$

Caso 3: $f(n)$ es mayor, la solución es $T(n) = \Theta(f(n))$

Caso 2: las dos funciones son del mismo tamaño, multiplicamos por un factor logarítmico, $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$

Implementación

```
long long karatsuba(long long x, long long y){
    int lenx, leny, n;
    long long a, b, c, d, p1, p2, p3;
    n = max(length(x), length(y));
    n = n + n % 2;
    lenx = n;
    leny = n;
    //cout<<"\nx:"<<bitset<8>(x)<<" y:"<<bitset<8>(y)<<"
    if (x==0 || y==0)
        return 0;
    if (x<10 && y<10)
        return x*y;
    else{
        a = x >> (lenx/2);
        b = (a << lenx/2) ^ x;
        c = y >> (leny/2);
        d = (c << leny/2) ^ y;
        //cout<<"\na:"<<bitset<8>(a)<<" b:"<<bitset<8>(b)<<"
        //cout<<"\nc:"<<bitset<8>(c)<<" d:"<<bitset<8>(d)<<"
        p1 = karatsuba(a, c);
        p2 = karatsuba(b, d);
        p3 = karatsuba(a+b, c+d);
        //cout<<"\n" << x << " * " << y << " = " << (2^n << n << " * " << p1 <<
        return pow(2, n) * p1 + pow(2, n/2) * (p3 - p2 - p1) + p2;
    }
}
```

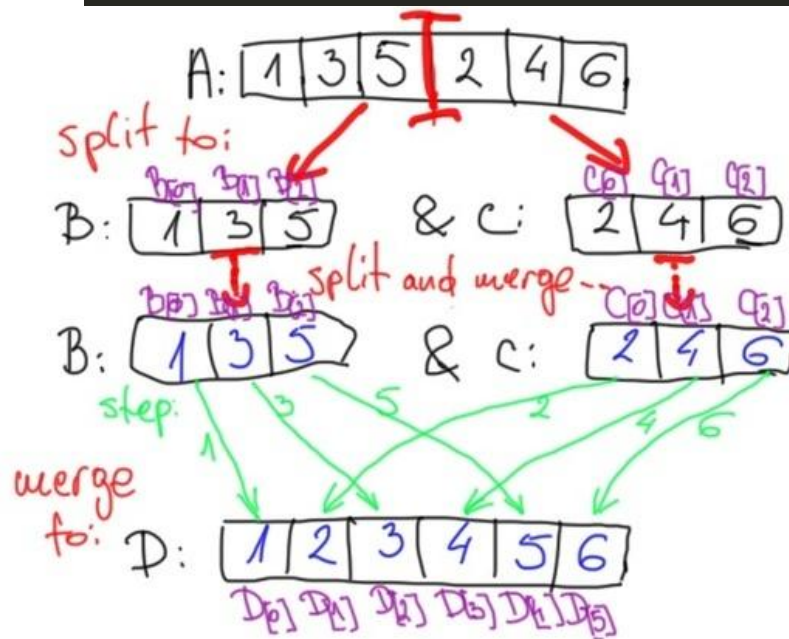
En este código la función Karatsuba son los dos números que va a recibir para llevar a cabo la multiplicación rápida, recibimos los números, obtenemos la longitud en bits de n que es mayor de entre los dos números, que es lo que nos permitirá determinar de dónde a donde vamos a correr (partir) los números por la mitad. Corremos a la izquierda el número correspondiente de veces $(n/2)$ el valor de X y de Y para obtener la mitad izquierda y derecha de cada número como en el ejemplo explicado en la teoría. Ahora solamente se manda a llamar recursivamente para calcular el valor de las multiplicaciones de $a*c$, $b*d$ y $(a+b)(c+d)$. Una vez que obtuvimos los valores de las multiplicaciones $p1, p2, p3$ solamente queda obtener el valor de la multiplicación total de $X*Y$ con la fórmula.

```

int mergeSort(int lower, int upper) {
    int count=0;
    if(lower < upper) {
        int middle = (lower + upper) / 2;
        count=mergeSort(lower,middle);
        count+=mergeSort(middle+1,upper);
        count+=merge(lower,middle,upper);
    }
    return count;
}

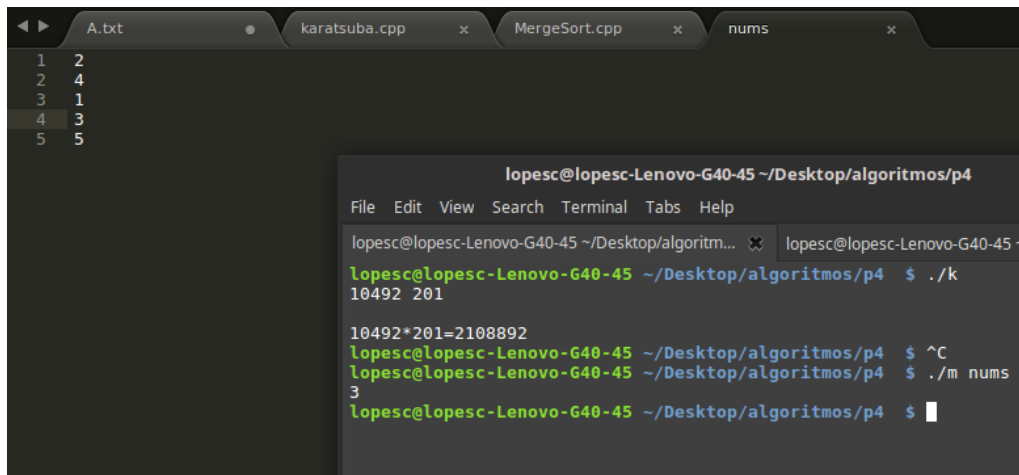
int merge(int lower, int middle ,int upper) {
    int inversion=0;
    queue <string> q1;
    queue <string> q2;
    for(int i=lower;i<=middle;i++) q1.push(v[i]);
    for(int i=middle+1;i<=upper;i++) q2.push(v[i]);
    int idx = lower;
    while(!q1.empty() && !q2.empty()) {
        if(q1.front() <= q2.front()) {
            v[idx++] = q1.front();
            q1.pop();
        } else {
            inversion=inversion+q1.size();
            v[idx++] = q2.front();
            q2.pop();
        }
    }
    while(!q1.empty()) {
        v[idx++] = q1.front();
        q1.pop();
    }
    while(!q2.empty()) {
        v[idx++] = q2.front();
        q2.pop();
    }
    return inversion;
}

```



Aquí la idea es prácticamente igual a la del mergesort original realizado en la práctica anterior la única diferencia es que cuando estemos ordenando las listas retornaremos el número de inversiones por cada merge. Tomando como ejemplo la imagen de arriba cuando el elemento de la lista B sea mayor al elemento de la lista C se suma a la variable count el total de elementos que hay en B, se hace esto ya que como sabemos los elementos en B y en C están ordenados, por ende si el elemento actual de B es mayor que C, los elementos restantes en B también serán mayores al elemento actual en C, ya solo queda hacer los pasos correspondientes al mergesort que sería hacer un pop en la lista B y hacer un Push en la lista D que es donde estará el merge entre B y C

Ejecución de programas



The screenshot shows a code editor with four tabs: 'A.txt', 'karatsuba.cpp', 'MergeSort.cpp', and 'nums'. The 'A.txt' tab is active, displaying a list of numbers: 1 2, 2 4, 3 1, 4 3, 5 5. Below the editor is a terminal window titled 'lopecsc@lopecsc-Lenovo-G40-45 ~/Desktop/algoritmos/p4'. The terminal shows the execution of a program with the following output:

```
lopecsc@lopecsc-Lenovo-G40-45 ~/Desktop/algoritmos/p4 $ ./k
10492 201
10492*201=2108892
lopecsc@lopecsc-Lenovo-G40-45 ~/Desktop/algoritmos/p4 $ ^C
lopecsc@lopecsc-Lenovo-G40-45 ~/Desktop/algoritmos/p4 $ ./m nums
3
lopecsc@lopecsc-Lenovo-G40-45 ~/Desktop/algoritmos/p4 $
```

Inversiones (2,1) (4,2)(4,3) haciendo un total de 3 usando mergesort.