Divide y vencerás

El concepto en la actualidad hace referencia a resolver un problema difícil dividiéndolo en partes mas simples tantas veces como sea necesario hasta que las soluciones de las partes se tornen obvia.

La solución del problema se construye con las soluciones encontradas.

PARADIGMA DyV:

Consiste en descomponer un problema en una serie de subproblemas de menor tamaño al resolver estos problemas usando siempre la misma técnica, las soluciones parciales se combinan para obtener la solución del problema original.

Los algoritmos DyV se expresan naturalmente mediante un algoritmo recursivo , ya que busca la resolución recursiva de un problema de manera que se divida en dos o mas subproblemas de igual tipo o similar.

Sin que se solapen.

El proceso continuo hasta que los subproblemas resultan ser lo suficientemente sencillos para que se resuelvan directamente y conquistar el problema. Al final las soluciones a cada uno de los subproblemas se combinan para dar solución al problema original.

Muchos algoritmos se basan en este paradigma como:

- Merge Sort
- Quick Sort
- Karatsuba
- Análisis Sintácticos
- La transformada de Fourier
- Strassen

Para que la solución DyV convenga debe cumplirse que:

1. Las operaciones descomponer y combinar deben de ser bastante eficientes de un orden de complejidad constante, logarítmico o lineal.
2. El numero de subproblemas generado debe ser pequeño [Típicamente dos subproblemas en cada partición].
3. Los subproblemas sean aprox. del mismo tamaño y no se solapen entre sí.

COMPLEJIDAD DEL PARADIGMA ALGORÍTMICO DYV:

La función complejidad para un problema de tamaño n es un sistema recurrente que suma:

- La complejidad de la división de los subproblemas.
- La complejidad de las llamadas recurrentes generalmente no lineales.
- La combinación de soluciones.

$$f(m) = f_{CalculaSoluciónInmediata}(m)$$

$$f(n) = f_{Divide}(n) + f_{Combina}(n) + \sum_{i=1}^{k} f(t[i])$$

El costo base de este modelo recurrente esta dado por el costo de la conquista al subproblema mas simple, generalmente de orden constante.

El diseño **Divide y Vencerás produce algoritmos recursivos** cuyo tiempo de ejecución se puede expresar mediante una ecuación en recurrencia del tipo:

$$T(n) = \begin{cases} cn^k, & si\ 1 \le n < b \\ aT(n/b) + cn^k, & si\ n \ge b \end{cases}$$

- $\boldsymbol{a, c\ y\ k}$ son números reales
- $\boldsymbol{n}$ y $\boldsymbol{b}$ son números naturales,
- $\boldsymbol{a > 0,\ c > 0,\ k \ge 0\ y\ b > 1}$
- $\boldsymbol{a}$ representa el número de subproblemas.
- $\boldsymbol{n/b}$ es el tamaño de cada uno de ellos.
- $\boldsymbol{cn^k}$ representa el coste de descomponer el problema inicial en los a subproblemas y el de combinar las soluciones para producir la solución del problema original, o bien el de resolver un problema elemental.

ORDENAMIENTO POR MEZCLA (MergeSort):

Este esquema de ordenamiento es un claro ejemplo de un DyV. Este ordenamiento busca dividir el problema hasta llegar al subproblema fácil, ordenar un arreglo de tamaño 1 [El cual ya está ordenado].

Por ello particiona el problema original en dos mitades las cuales recursivamente se irán partiendo y dividiendo, hasta que se llegue al problema fácil; una vez que se obtenga, se irán devolviendo las respuestas de ambas mitades, las cuales se combinando, guardando en un nuevo arreglo para ser retornado en la llamada anterior. Esta combinación sucederá en cada retorno de nivel recursivo hasta alcanzar la llama inicial. Esto tiene un costo constante en cada división que se realizada y debido a que cada que se realiza al problema se le hace una partición en dos hasta llegar al caso base el árbol de recursión crece una altura:

$$\log_2 \text{n-1}$$

Por otro lado, cada recursión realiza una combinación en orden lineal por lo que el algoritmo logra realizar el ordenamiento en orden:

$$\text{n log(n)}$$

Para todos sus casos.

-Complejidad temporal-

$$\text{T(n)} = \begin{cases} 1 & \text{si } 1 = 1 \\ 2\text{T}\ (\text{n}/2) + \text{n} & \text{si n} > 2 \end{cases} \qquad \text{T(n)}\ \epsilon\ \text{O(n log n)}$$

ORDENAMIENTO RAPIDO (QuickSort)

Este esquema de ordenamiento tambien es un algoritmo DyV, pero a diferencia del algoritmo de mezcla. La subdivisión de los problemas requiere un proceso de intercambio en torno a un elemento llamado pivote antes de llamar recursivamente a la siguiente división del problema.

Esta operación de dividir realiza en orden lineal un intercambio de elementos en el arreglo entono al valor del elemento pivote, posteriormente el árbol de recursión avanza hasta llegar al caso simple nuevamente el arreglo de un número, pero en el proceso de pivoteo crear las

particione del problema entorno al pivote se asegura que el número pivote ya quedara ordenado. Por lo que la combinación para el ordenamiento ya se ha realizado de forma implícita y no requiere realizarse ninguna operación al retornar la recursión.

El calculo de la complejidad de este ordenamiento no es tan trivial debido a que en cada división se desconoce el tamaño de los subproblemas que se formaran, pudiendo en su peor caso se muy desbalanceado en torno al pivote, lo que puede hacer que la complejidad del algoritmo en su peor caso sea de orden cuadrático.

Matemáticamente la probabilidad de que esto suceda es muy baja y puede realizarse un analisis matemático de este algoritmo encontrando que la probabilidad de que este ordenamiento tenga costo logarítmico es muy alta, logrando concluir que:

En el caso medio este algoritmo es de orden n*log(n)

Este algoritmo es mas usado que Merge pese a su peor caso ya que:

Una implementación adecuada utiliza menos memoria

que el ordenamiento por mezcla

Además, que pueden beneficiarse de la memoria cache del procesador de su ejecución.

-Complejidad temporal-

$$T(n) \epsilon O(n^2)$$

$$T(n) \epsilon \Theta(n \log_2 n)$$

$$T(n) \epsilon \Omega(n \log_2 n)$$

BUSQUEDA DE MAXIMO Y MINIMO:

Para este caso en particular, al usar el paradigma DyV nos arroja la misma complejidad que su solución en fuerza bruta. Sin embargo, la solución DyV nos permite paralelizar su solución.

```
MaxMinDV (A: array [1..N] of tipo; i, j: integer; var Max, Min: tipo)
        si i<j-1
        /*Dividir el problema en 2 subproblemas*/
                mit = (i+j) div 2
                MaxMinDV (i, mit, Max1, Min1)
                MaxMinDV (mit+1, j, Max2, Min2)
        /*Combinar*/
                si Max1>Max2
                                Max= Max1
                en otro caso
                                Max = Max2
                si Min1<Min2
                        Min = Min1
                en otro caso
                        Min = Min2
        /*Caso base*/
        en otro caso si i=j-1
                si A[i]>A[j]
                        Max = A[i] ; Min = A[j]
                en otro caso
                        Max = A[j] ; Min= A[i]
        en otro caso
                Max = A[i] ; Min = Max
```

$$T(n) = \begin{cases} 2 & \text{si } 1 \leq n \leq 2 \\ 2T(n/2) + 2 & \text{si } n > 2 \end{cases}$$

$$T(n) \epsilon O(n)$$

DISEÑO DE SOLUCIONES DYV:

No existe una metodología a ser seguida para crear una solución bajo este paradigma, cada problema se debe analizar de forma **TOP DOWN** de arriba hacia abajo.

¿Cómo dividir en bajo orden de complejidad?

¿Cuál es la conquista fácil?

Y una vez que esta conquista es alcanzada ¿Cómo resolver la combinación de las soluciones?

La combinación requerida para resolver el problema original es el pensamiento **BOTTOM DOWN** de abajo hacia arriba, que se debe hacer sobre el problema.

En ciertos diseños DyV podemos tener una combinación que esta implícita desde el pensamiento **TOP DOWN** por la forma en que el problema se dividió que podrá quizás no ser necesaria una combinación de forma explicita como es el caso del ordenamiento rápido.

La complejidad que tiene un algoritmo DyV, puede mejorar otro esquema de solución a un problema bajarlo a un orden lineal, logarítmico o nlog(n).

O puede empeorarlo por lo que no se debe tomar a la ligera.

OBSERVACIONES:

1. Si el algoritmo solo tiene una llamada recursiva, es decir k=1, hablamos de algoritmos de simplificación. (Ejemplo: Búsqueda binaria)
2. Para el DyV es importante que los subproblemas sean independientes (No solapamiento)
3. La división de los subproblemas debe ser lo más equilibrada posible (Repartir la carga)
4. DyV permite paralelizar una solución en problemas parciales e independientes.

CONCLUSIÓN:

Un algoritmo DyV es una alternativa de solución, típicamente usando recursión y permite de manera natural resolver subproblemas que no se solapan en un proceso recursivo único. Este esquema puede no mejorar la complejidad de otras soluciones, pero posee una ventaja por que permite paralelizar una solución a problemas parciales e independientes y se utiliza un procesamiento concurrente o paralelo, probablemente usar este esquema tenga una gran ventaja.