GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: DIVIDE Y VENCERÁS

1. Introducción

El término Divide y Vencerás en su acepción más amplia es algo más que una técnica de diseño de algoritmos. De hecho, suele ser considerada una filosofía general para resolver problemas y de aquí que su nombre no sólo forme parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.

2. Conocimientos previos

2.1. Recursividad

Recursividad es, en ciencias de la computación, una forma de atajar y solventar problemas. De hecho, recursión es una de las ideas centrales de ciencia de computación. Resolver un problema mediante recursión significa que la solución depende de las soluciones de pequeñas instancias del mismo problema. Un algoritmo recursivo es un algoritmo que expresa la solución de un problema en términos de una llamada a sí mismo. La llamada a sí mismo se conoce como llamada recursiva o recursión.

3. Desarrollo

En nuestro contexto, Divide y Vencerás es una técnica de diseño de algoritmos que consiste en resolver un problema a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño. Si los subproblemas son todavía relativamente grandes se aplicará de nuevo esta técnica hasta alcanzar subproblemas lo suficientemente pequeños para ser solucionados directamente. Ello naturalmente sugiere el uso de la recursión en las implementaciones de estos algoritmos.

La resolución de un problema mediante esta técnica consta fundamentalmente de los siguientes pasos:

1. En primer lugar ha de plantearse el problema de forma que pueda ser descompuesto en k subproblemas del mismo tipo, pero de menor tamaño. Es decir, si el tamaño de la entrada es n, hemos de conseguir dividir el problema en k subproblemas (donde $1 \leq k \leq n$) cada uno con una entrada de tamaño n_k y donde $0 \leq n_k \leq n$. A esta tarea se le conoce como *división*.
2. En segundo lugar han de resolverse independientemente todos los subproblemas, bien directamente si son elementales o bien de forma recursiva. El hecho de que el tamaño de los subproblemas sea estrictamente menor que el tamaño original del problema nos garantiza la convergencia hacia los casos elementales, también denominados casos *base*.
3. Por último, *combinar* las soluciones obtenidas en el paso anterior para construir la solución del problema original.

Hemos de hacer unas apreciaciones en este esquema sobre el procedimiento Divide, sobre el número k que representa el número de subproblemas, y sobre el tamaño de los subproblemas, ya que de todo ello va a depender la eficiencia del algoritmo resultante.

En primer lugar, el número k debe ser pequeño e independiente de una entrada determinada. En el caso particular de los algoritmos Divide y Vencerás que contienen sólo una llamada recursiva, es decir $k = 1$, hablaremos de algoritmos de *simplificación*. Tal es el caso del algoritmo recursivo que resuelve el cálculo del factorial de un número, que sencillamente reduce el problema a otro subproblema del mismo tipo de tamaño más pequeño. También son algoritmos de simplificación el de búsqueda binaria en un vector o el que resuelve el problema del k-ésimo elemento.

La ventaja de los algoritmos de simplificación es que consiguen reducir el tamaño del problema en cada paso, por lo que sus tiempos de ejecución suelen ser muy buenos (normalmente de orden logarítmico o lineal). Además pueden admitir una mejora adicional, puesto que en ellos suele poder eliminarse fácilmente la recursión mediante el uso de un bucle iterativo, lo que conlleva menores tiempos de ejecución y menor complejidad espacial al no utilizar la pila de recursión, aunque por contra, también en detrimento de la legibilidad del código resultante.

Por el hecho de usar un diseño recursivo, los algoritmos diseñados mediante la técnica de Divide y Vencerás van a heredar las ventajas e inconvenientes que la recursión plantea:

- Por un lado el diseño que se obtiene suele ser simple, claro, robusto y elegante, lo que da lugar a una mayor legibilidad y facilidad de depuración y mantenimiento del código obtenido.
- Sin embargo, los diseños recursivos conllevan normalmente un mayor tiempo de ejecución que los iterativos, además de la complejidad espacial que puede representar el uso de la pila de recursión.

Desde un punto de vista de la eficiencia de los algoritmos Divide y Vencerás, es muy importante conseguir que los subproblemas sean independientes, es decir, que no exista solapamiento entre ellos. De lo contrario el tiempo de ejecución de estos algoritmos será exponencial. Como ejemplo pensemos en el cálculo de la sucesión de Fibonacci, el cual, a pesar de ajustarse al esquema general y de tener sólo dos llamadas recursivas, tan sólo se puede considerar un algoritmo recursivo pero no clasificarlo como diseño Divide y Vencerás. Esta técnica está concebida para resolver problemas de manera eficiente y evidentemente este algoritmo, con tiempo de ejecución exponencial, no lo es.

En cuanto a la eficiencia hay que tener en también en consideración un factor importante durante el diseño del algoritmo: el número de subproblemas y su tamaño, pues esto influye de forma notable en la complejidad del algoritmo resultante.

Otra consideración importante a la hora de diseñar algoritmos Divide y Vencerás es el reparto de la carga entre los subproblemas, puesto que es importante que la división en subproblemas se haga de la forma más equilibrada posible. En caso contrario nos podemos encontrar con *anomalías de funcionamiento* como le ocurre al algoritmo de ordenación *Quicksort*. Éste es un representante claro de los algoritmos Divide y Vencerás, y su caso peor aparece cuando existe un desequilibrio total en los subproblemas al descomponer el vector original en dos subvectores de tamaño 0 y $n - 1$. En este caso su orden es O(n^2), frente a la buena complejidad, O($n \log n$), que consigue cuando descompone el vector en dos subvectores de igual tamaño. También es interesante tener presente la dificultad y es esfuerzo requerido en cada una de estas fases va a depender del plan-

teamiento del algoritmo concreto. Por ejemplo, los métodos de ordenación por Mezcla y Quicksort son dos representantes claros de esta técnica pues ambos están diseñados siguiendo el esquema presentado: dividir y combinar.

4. Implementación

El funcionamiento de los algoritmos que siguen la técnica de Divide y Vencerás descrita anteriormente se refleja en el esquema general que presentamos a continuación:

```
PROCEDURE DyV(x:TipoProblema):TipoSolucion;
   VAR i,k,:CARDINAL;
      s:TipoSolucion;
      subproblemas: ARRAY OF TipoProblema;
      subsoluciones:ARRAY OF TipoSolucion;
BEGIN
   IF EsCasobase(x) THEN
      s:=ResuelveCasoBase(x)
   ELSE
      k:=Divide(x,subproblemas);
      FOR i:=1 TO k DO
         subsoluciones[i]:=DyV(subproblemas[i])
      END;
      s:=Combina(subsoluciones)
   END;
   RETURN s
END DyV;
```

5. Complejidad

En definitiva, el diseño Divide y Vencerás produce algoritmos recursivos cuyo tiempo de ejecución se puede expresar mediante una ecuación en recurrencia del tipo:

$$T(n) = \begin{cases} cn^k & \text{si } 1 \leq n < b \\ aT(n/b) + cn^k & \text{s1 } n \geq b \end{cases}$$

donde a, c y k son números reales, n y b son números naturales, y donde $a > 0$, $c > 0$, $k \geq 0$ y $b \geq 1$. El valor de a representa el número de subproblemas, n/b es el tamaño de cada uno de ellos, y la expresión 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. La solución a esta ecuación es:

$$T(n) \in \begin{cases} \theta(n^k) & \text{si } a < b^k \\ \theta(n^k \log n) & \text{si } a = b^k \\ \theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Las diferencias surgen de los distintos valores que pueden tomar a y b, que en definitiva determinan el número de subproblemas y su tamaño. Lo importante es observar que en todos los casos la complejidad es de orden polinómico o polilogarítmico pero nunca exponencial, frente a los algoritmos recursivos que pueden alcanzar esta complejidad en muchos casos. Esto se debe normalmente a la repetición de los cálculos que se produce al existir solapamiento en los subproblemas en los que se descompone el problema original.

Para aquellos problemas en los que la solución haya de construirse a partir de las soluciones de subproblemas entre los que se produzca necesariamente solapamiento existe otra técnica de diseño más apropiada, y que permite eliminar el problema de la complejidad exponencial debida a la repetición de cálculos. Estamos hablando de la *Programación Dinámica*.

6. Aplicaciones

La uso de esta técnica para el diseño de algoritmo ha posibilitado que podamos contar con algoritmos ya bien definidos que se apoyan en esta técnica o basados en esta técnica para resolver problemas concretos o se pueden implementar algoritmos capaces de solucionar determinados problemas cuyas soluciones triviales presentan mayor complejidad que los algoritmos diseñados bajo esta técnica. A continuación una lista de estos ejemplos:

1. Búsqueda binaria, binaria no centrada y ternaria son ejemplos claros de la técnica Divide y Vencerás. El problema de partida es decidir si existe un elemento dado x en un vector de enteros ordenado. El hecho de que esté ordenado va a permitir utilizar esta técnica
2. Para mulplicar u y v que son dos números naturales de n bits donde, por simplicidad, n es una potencia de 2.
3. Buscar la moda en un arreglo.
4. Buscar la mediana de dos arreglos
5. La multiplicación de matrices cuadradas
6. Buscar la subsecuencia de suma máxima de elementos concecutivos en un arreglo.
7. Organizar el pareo de un torneo con n jugadores en donde cada jugador ha de jugar exactamente una vez contra cada uno de sus posibles $n-1$ competidores, y además ha de jugar un partido cada día,teniendo a lo sumo un día de descanso en todo el torneo.
8. El elemento en su posición. Sea $a[1 \ldots n]$ un arreglo ordenado de enteros todos distintos. Nuestro problema es implementar un algoritmo de complejidad O(logn) en el peor caso capaz de encontrar un índice i tal que $1 \leq i \leq n$ y $a[i] = i$, suponiendo que tal índice exista.

9. El elemento mayoritario de un arreglo. Sea $a[1 \ldots n]$ un arreglo de enteros. Un elemento x se denomina elemento mayoritario de a si x aparece en el vector más de $n/2$ veces,