Algoritmia/Programación dinámica

La **programación dinámica** puede verse como una optimización de otros métodos algorítmicos (../Divide y vencerás/, ../Vuelta atrás/, ...) en los que una solución a un problema puede implicar la repetición continua de subproblemas. En concreto, la programación dinámica suele implicar la utilización de una tabla auxiliar donde almacenar las soluciones a los subproblemas ya calculados, de tal forma que cada subsolución se calcule una única vez, reduciendo así el coste en tiempo a cambio de coste en espacio (almacenamiento en memoria de la tabla).

El término dinámico proviene de que la tabla auxiliar se rellena en tiempo de ejecución. No debe confundirse la programación dinámica en el ámbito algorítmico con los lenguajes de programación dinámicos, como Scheme o Lisp.

Ejemplo: la sucesión de Fibonacci

La primera implementación de una función para encontrar el **n**-ésimo término de la sucesión de Fibonacci que surge de forma natural, es la basada en la definición matemática de la misma:

```
fun fib(n)
    si n = 0 || n = 1
        devolver n
    si no
        devolver fib(n - 1) + fib(n - 2)
    fsi
ffun
```

No es difícil comprobar la cantidad de cálculo redundante a que conduce este primer acercamiento al problema. Si llamamos, por ejemplo, a `fib (5)`, produciremos un árbol de llamadas que contendrá funciones con los mismos parámetros varias veces:

1. `fib(5)`
2. `fib(4) + fib(3)`
3. `(fib(3) + fib(2)) + (fib(2) + fib(1))`
4. `((fib(2) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))`
5. `(((fib(1) + fib(0)) + fib(1)) + (fib(1) + fib(0))) + ((fib(1) + fib(0)) + fib(1))`

En particular, `fib(2)` ha sido calculado dos veces desde cero. En ejemplos mayores, muchos otros valores de `fib`, o subproblemas, son recalculados, llevando a un algoritmo de complejidad exponencial.

Si se considera ahora la utilización de un simple array para almacenar la solución a los distintos subproblemas, la función resultante será de complejidad lineal, en lugar de exponencial, pues cada `fib(n)` se calcula una única vez:

```
var tabla : array [0..n] de ent
fun fib(n)
   si tabla[n] = -1 entonces                // El array inicializado a -1
       tabla[n] := fib(n - 1) + fib(n - 2)
   fsi
   devolver tabla[n]
ffun
```

Esta implementación está hecha desde arriba (*top-down* en inglés), solicitando primero la solución del problema y calculando recursivamente las soluciones a los subproblemas. Una implementación desde abajo (*bottom-up* en

inglés) consiste en realizar el proceso inverso: calcular primero la solución a los subproblemas y a partir de las mismas, definir la solución al problema.

Problemas resueltos

Ejecución de n tareas en tiempo mínimo

http:/ / es. wikipedia. org/ wiki/ Ejecución_de_n_tareas_en_tiempo_mínimo_en_un_sistema_de_dos_procesadores_A_y_B

Problema de los sellos

http://es.wikipedia.org/wiki/Problema_de_los_sellos_con_programación_dinámica

Problema de la mochila

http://es.wikipedia.org/wiki/Problema_de_la_mochila_con_programación_dinámica

Problema del producto de una secuencia de matrices

http:/ / es. wikipedia. org/ wiki/ Problema_del_producto_de_una_secuencia_de_matrices_con_programación_dinámica

Problema de las monedas

http://es.wikipedia.org/wiki/Problema_de_las_monedas_con_programación_dinámica

Problema del coste mínimo entre dos nodos de un grafo dirigido

http://es.wikipedia.org/wiki/Camino_de_coste_mínimo_entre_dos_nodos_de_un_grafo_dirigido

Problema de la división de peso

http://es.wikipedia.org/wiki/Problema_de_la_división_de_peso

Problema de las vacas

http://es.wikipedia.org/wiki/Problema_de_las_vacas_con_programación_dinamica

Fuentes y contribuyentes del artículo

Algoritmia/Programación dinámica *Fuente*: http://es.wikibooks.org/w/index.php?oldid=44215 *Contribuyentes*: Gothmog

Licencia