***Tecnica de Programacion Dinamica***

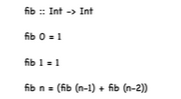
Una forma general de describir la estrategia Programación Dinámica es la siguiente:

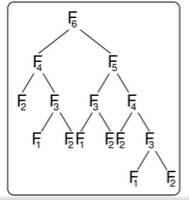
Dado un problema P, para el cual:

* Ya se cuenta con una solución Divide & Conquer (top-down, en el sentido que para resolver el problema lo divide en subproblemas más pequeños, y combina sus soluciones para resolver el problema original), PERO los subproblemas a resolver NO son independientes, sino que comparten a su vez subproblemas (es decir, se “solapan”), y por lo tanto requieren recomputar varias veces las soluciones a algunos subproblemas comunes.
* Resolver el problema mediante Programación Dinámica consiste en invertir el orden de cómputo de soluciones a subproblemas de P (es decir, computarlas bottom-up), y almacenar los resultados de manera tal de no tener que recomputarlos.
* Programación Dinámica cambia eficiencia por espacio de almacenamiento, pero en muchos

casos este intercambio es extremadamente beneficioso.

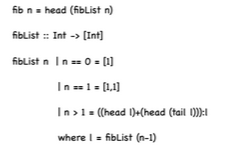
***Ejemplo : Números de Fibonacci***

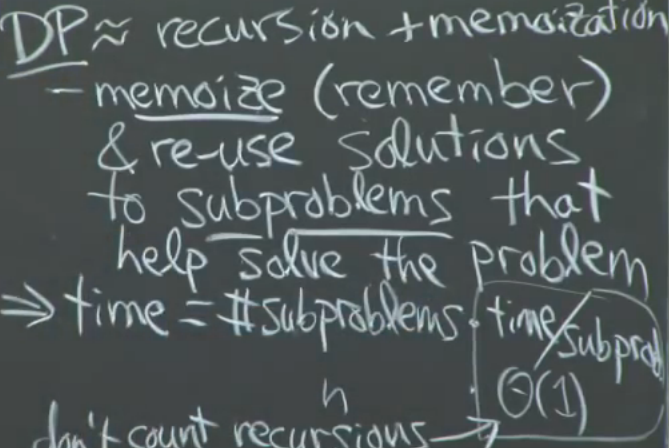


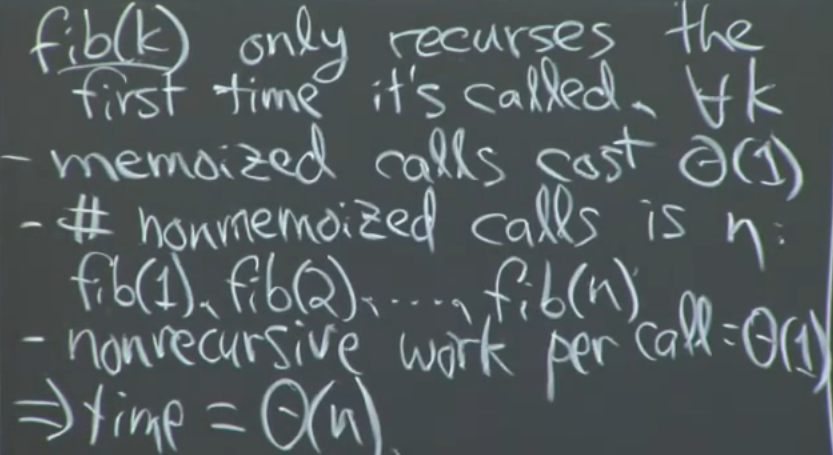


*Decir que un problema tiene subproblemas superpuestos es decir que se usa un mismo subproblema para resolver diferentes problemas mayores. Por ejemplo, en la sucesión de Fibonacci (F3 = F1 + F2 y F4 = F2 + F3) calcular cada término supone calcular F2. Como para calcular F5 hacen falta tanto F3 como F4, una mala implementación para calcular F5 acabará calculando F2 dos o más veces. Esto sucede siempre que haya subproblemas superpuestos: una mala implementación puede acabar desperdiciando tiempo recalculando las soluciones óptimas a problemas que ya han sido resueltos anteriormente.*

**Esto se puede evitar guardando las soluciones que ya hemos calculado. Entonces, si necesitamos resolver el mismo problema más tarde, podemos obtener la solución de la lista de soluciones calculadas y reutilizarla.**







Para poder encarar una solución a un problema mediante Programación Dinámica debe contarse antes con una solución recursiva/Divide & conquer para el problema.

La PD es un ejemplo importante de intercambio de eficiencia en tiempo por eficiencia en espacio.

En casos en que el almacenamiento de soluciones a subproblemas es realizado sobre una estructura del estilo de arreglos, lista o matrices, debe asegurarse que el “poblado ” de la estructura con valores se haga en un orden adecuado: al computar un valor dado ya deben haberse computado los valores correspondientes a los subproblemas del problema corriente.