EP de Ingeniería de Software

Programación Dinámica

Análisis y Diseño de Algoritmos, 2024-1

Jorge Luis Chávez Soto

Contenido

Introducción

- Estas técnicas están diseñadas para aplicarse sobre determinados problemas.
- Cualquier fórmula matemática recursiva se puede traducir directamente a un algoritmo recursivo, pero quizá éste no necesariamente será eficiente.
- Una solución a este problema es escribir un algoritmo que ***"guarde respuestas"*** ó ***"soluciones parciales"*** en una tabla.
- Esta técnica se denomina ***"programación dinámica"***.

Definición de la Programación Dinámica

- Los algoritmos resuelven problemas **descomponiéndolos** en varios subproblemas que se resuelven recursivamente.
- Los resultados de los subproblemas mas pequeños son **reutilizados** en el calculo de subproblemas más grandes.
- La programación dinámica es una **técnica ascendente** que comienza habitualmente resolviendo los problemas más pequeños, guardando estos resultados.
- Entonces reutilizándolos para resolver subproblemas cada vez mayores hasta que se obtiene una solución al problema original.

Definición de la Programación Dinámica

- Un planteamiento de programación dinámica solo se justifica si existe un cierto grado de **solapamiento** en los subproblemas.

- La idea es **evitar el calculo** del mismo resultado dos veces. Esto se consigue construyendo una tabla en memoria, y rellanándola con resultados obtenidos conforme se van calculando.

- Téngase en cuenta que recuperar un resultado dado de esta tabla requiere un tiempo Q(1).

Método General.

- La **programación dinámica** se basa en el **razonamiento inductivo.** La misma idea del **divide y vencerás,** pero aplicando una estrategia distinta.

- **Similitud:**
  - Descomposición recursiva del problema.
  - Se obtiene aplicando un razonamiento inductivo.

- **Diferencia:**
  - Divide y vencerás: aplicar directamente la fórmula recursiva.

  - Programación dinámica: resolver los problemas más pequeños, guardando los resultados en una tabla (programa iterativo).

Caminos Mínimos (Algoritmo de Floyd)

- Para calcular los caminos mínimos entre cualquier par de nodos de un grafo.

- **Razonamiento inductivo:** para calcular los caminos mínimos pudiendo pasar por los **k** primeros nodos usamos los caminos mínimos pasando por los **k-1** primeros.

- $D_k(i, j)$**:** camino mínimo de **i** a **j** pudiendo pasar por los nodos 1, 2, …, **k**.

$$D_k(i,j) = \begin{cases} C[i,j] & si\ K = 0 \\ \min(D_{K-1}[i,j]\ , D_{K-1}[i,k] + D_{K-1}[k,j]) & si\ K > 0 \end{cases}$$

- $D_n(i, j) \rightarrow$ caminos mínimos finales

Caminos Mínimos (Algoritmo de Floyd)

- Aplicación de la fórmula:
  - Empezar por el problema pequeño: **k** = 0
  - Avanzar hacia problemas más grandes: **k** = 1, 2, 3, ...

- ¿Cómo se garantiza que un algoritmo de programación dinámica obtiene la solución correcta?

- Una descomposición es correcta si cumple el

- **Principio de optimalidad de Bellman**:

- La solución óptima de un problema se obtiene combinando soluciones óptimas de subproblemas.

Caminos Mínimos (Algoritmo de Floyd)

- O bien: cualquier subsecuencia de una secuencia óptima debe ser, a su vez, una secuencia óptima.

- **Ejemplo.** Si el camino mínimo de **A** a **B** pasa por **C**, entonces los trozos de camino de **A** a **C**, y de **C** a **B** deben ser también mínimos.

- **Ojo:** el principio no siempre es aplicable.

- **Contraejemplo.** Si el camino simple más largo de **A** a **B** pasa por **C**, los trozos de **A** a **C** y de **C** a **B** no tienen por qué ser soluciones óptimas.

Pasos para aplicar programación dinámica

1) Obtener una **descomposición recurrente** del problema:
   - Ecuación recurrente.
   - Casos base.

2) Definir la **estrategia** de aplicación de la fórmula:
   - Tablas utilizadas por el algoritmo.
   - Orden y forma de rellenarlas.

3) Especificar cómo se **recompone la solución** final a partir de los valores de las tablas.

- **Punto clave:** obtener la descomposición recurrente.
- Requiere mucha "creatividad"...

Pasos para aplicar programación dinámica

- Cuestiones a resolver en el razonamiento inductivo:
  - ¿Cómo reducir un problema a subproblemas más simples?
  - ¿Qué parámetros determinan el **tamaño del problema** (es decir, cuándo el problema es "más simple")?

- **Idea:** ver lo que ocurre al tomar una decisión concreta → interpretar el problema como un proceso de toma de decisiones.

- **Ejemplos. Floyd.** Decisiones: Pasar o no pasar por un nodo intermedio.

- **Mochila 0/1.** Decisiones: coger o no coger un objeto dado.

Análisis de tiempos de ejecución.

- La programación dinámica se basa en el uso de **tablas** donde se almacenan los resultados parciales.

- En general, el **tiempo** será de la forma:

  Tamaño de la tabla * Tiempo de rellenar cada elemento de la tabla.

- Un aspecto **importante** es la memoria puede llegar a ocupar la tabla.

- Además, algunos de estos cálculos pueden ser innecesarios.

Métodos ascendentes y descendentes

- **Métodos descendentes (divide y vencerás)**
  - Empezar con el problema original y descomponer recursivamente en problemas de menor tamaño.
  - Partiendo del problema grande, descendemos hacia problemas más sencillos.
- **Métodos ascendentes (programación dinámica)**
  - Resolvemos primero los problemas pequeños (guardando las soluciones en una tabla). Después los vamos combinando para resolver los problemas más grandes.
  - Partiendo de los problemas pequeños avanzamos hacia los más grandes.

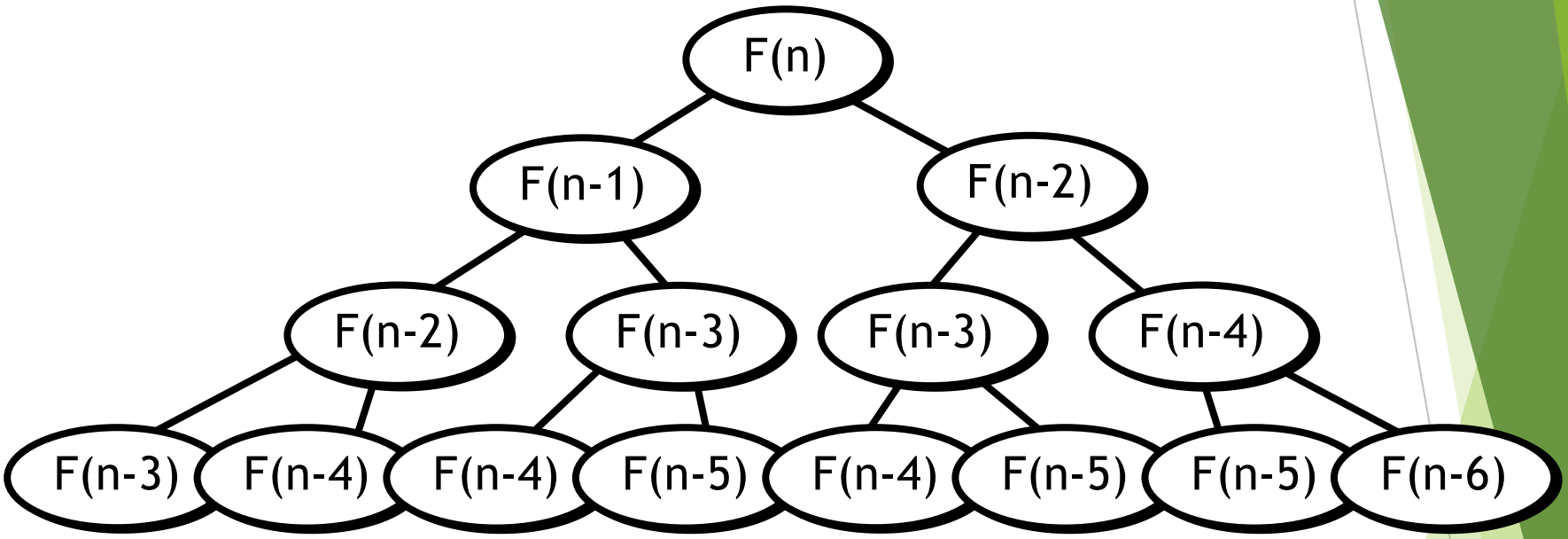
Problema de Fibonacci

- Podemos dar una nueva versión de la función que calcula el n-ésimo término de la sucesión de Fibonacci.

$$F(n) = \begin{cases} 1 & \text{Si } n = 1 \text{ ó } n = 0 \\ F(n-1) + F(n-2) & \text{Si } n > 1 \end{cases}$$

- Solucionar esto usando fuerza bruta que es el algoritmo recursivo, obliga a cálculos repetidos, F(2) es calculado dos veces para obtener F(4).

Problema de Fibonacci



- **Problema:** Muchos cálculos están repetidos.
- **Con divide y vencerás:** : $\Theta(1{,}62^n)$
- **Con programación dinámica:** $\Theta(n)$.

Problema de Fibonacci

- La programación dinámica evita recalcular los valores intermedios almacenándolos en una tabla.
  - Un arreglo F(0 ... n) conteniendo n+1 espacios y el valor de F(i) es almacenado en F(i).
  - Empezamos guardando F(0) y F(1); entonces, F(i) se calcula sumando F(i – 2) y F(i – 1) para i = 1, 2, 3 ..., n.
  - Como F(i – 2) y F(i – 1) pueden ser recuperados de la tabla y sumados en tiempo constante.
- Esta vez, utilizaremos un vector para no calcular varias veces el mismo término de la sucesión.

Conclusiones

- El **razonamiento inductivo** es una herramienta muy potente en resolución de problemas.
- Aplicable no sólo en problemas de optimización.
- ¿Cómo obtener la fórmula? Interpretar el problema como una serie de **toma de decisiones**.
- Descomposición recursiva no necesariamente implica implementación recursiva.
- **Programación dinámica:** almacenar los resultados en una tabla, empezando por los tamaños pequeños y avanzando hacia los más grandes.

Gracias

Jorge Luis Chávez Soto

jchavezs@unmsm.edu.pe

https://twitter.com/jlchavezs

https://bit.ly/3YvxhWN