Análisis de Algoritmos
Unidad 4: Programacion Dinamica

Ing. Matias Valdenegro T.

Universidad Tecnológica Metropolitana

22 de junio de 2012

Paradigma “Divide y Venceras”

Teorema Maestro

Programacion Dinamica

Paradigma “Divide y Venceras”

Teorema Maestro

Programacion Dinamica

“Divide y Venceras”

Es un paradigma de diseño de algoritmos en el cual la idea es dividir un problema en sub-problemas mas pequeños, y usar la solucion de dichos sub-problemas para construir la solucion del problema original.

Estructura de un Algoritmo D&V

1. Dividir el problema en sub-problemas mas pequeños.
2. Solucionar los sub-problemas pequeños.
3. Combinar las soluciones a los sub-problemas para obtener la solucion del problema original.

Estructura de un Algoritmo D&V

Si nuestro algoritmo D&V toma un problema de largo N, lo divide en A sub-problemas de largo B, y requiere un costo $f(n)$ para combinar las soluciones, entonces el costo $T(n)$ del algoritmo es:

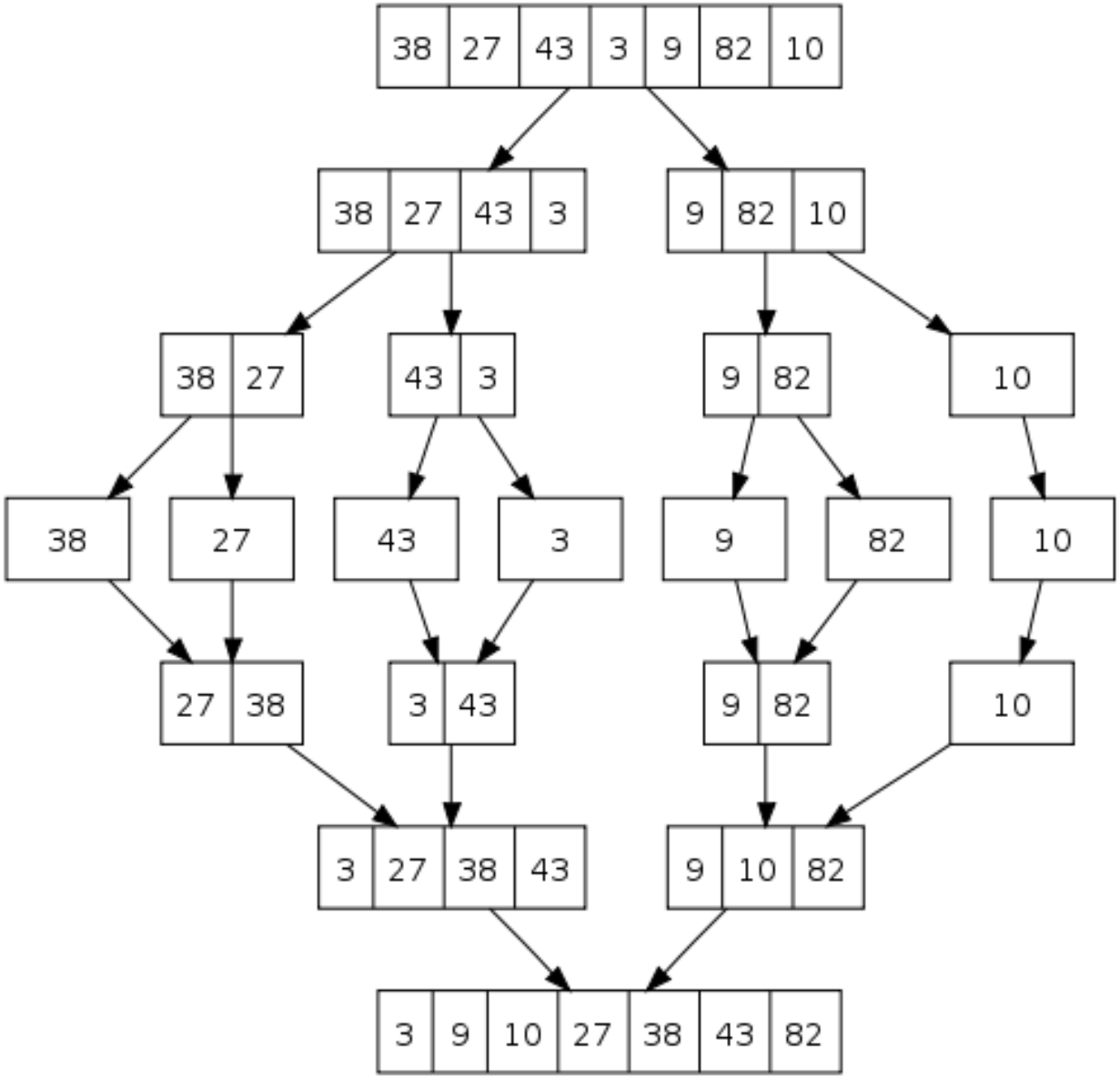
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Ejemplo: Ordenamiento

Mergesort es un algoritmo de ordenamiento que usa el paradigma divide-y-venceras. Dado un arreglo A de N elementos:

- Dividir: Particionar A en 2 arreglos A_1 y A_2, de $\frac{n}{2}$ elementos cada uno.
- Recursividad: Ordenar recursivamente A_1 y A_2.
- Combinar: Fusionar (Merge) ambos arreglos ordenados A_1 y A_2 en un unico arreglo ordenado.

Ejemplo: Ordenamiento



Analisis Ejemplo

Podemos plantear la ecuacion de recurrencia que represente el costo $T(n)$ de Mergesort:

$$T(1) = c$$
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Cual es la solucion de esta ecuacion?

Analisis Ejemplo

Podemos plantear la ecuacion de recurrencia que represente el costo $T(n)$ de Mergesort:

$$T(1) = c$$
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Cual es la solucion de esta ecuacion? $T(n) \in O(n \log n)$.

Ejemplo: Multiplicacion de Matrices

En la asignatura de Algebra se ve el algoritmo clasico de multiplicacion para $C = A \times B$:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

Este algoritmo tiene complejidad $O(n^3)$, donde A, B y C son matrices de $N \times N$. Es posible mejorar esto?

Ejemplo: Multiplicacion de Matrices

Supongamos que dividimos A, B y C en sub-matrices de $\frac{n}{2} \times \frac{n}{2}$ cada una:

$$\begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \times \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

Entonces:

$$\begin{aligned} C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\ C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\ C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\ C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2} \end{aligned}$$

Analisis Ejemplo

Para calcular el producto de matrices de $N \times N$, debemos calcular 8 productos de matrices de $\frac{n}{2} \times \frac{n}{2}$, seguido de 4 sumas (Las cuales tienen un costo $O(n^2)$). Entonces, la ecuacion de recurrencia para el costo $T(n)$ es:

$$T(1) = c$$
$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

Cual es la solucion de dicha ecuacion?

Analisis Ejemplo

Para calcular el producto de matrices de $N \times N$, debemos calcular 8 productos de matrices de $\frac{n}{2} \times \frac{n}{2}$, seguido de 4 sumas (Las cuales tienen un costo $O(n^2)$). Entonces, la ecuacion de recurrencia para el costo $T(n)$ es:

$$T(1) = c$$
$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

Cual es la solucion de dicha ecuacion? $T(n) \in O(n^3)$. Esto no es una mejora, tiene la misma complejidad que el algoritmo trivial.

Ejemplo: Multiplicacion de Matrices 2.0

Strassen propuso dividir las matrices de la misma forma, pero calcular el producto de las submatrices de diferente forma:

$$
\begin{aligned}
M_0 &= (A_{1,1} + A_{2,2}) \times (B_{1,1} + B_{2,2}) \\
M_1 &= (A_{1,2} - A_{2,2}) \times (B_{2,1} + B_{2,2}) \\
M_2 &= (A_{1,1} - A_{2,1}) \times (B_{1,1} + B_{1,2}) \\
M_3 &= (A_{1,1} + A_{1,2}) \times B_{2,2}) \\
M_4 &= A_{1,1} \times (B_{1,2} - B_{2,2}) \\
M_5 &= A_{2,2} \times (B_{2,1} - B_{1,1}) \\
M_6 &= (A_{2,1} + A_{2,2}) \times B_{1,1}
\end{aligned}
$$

$$
\begin{aligned}
C_{1,1} &= M_0 + M_1 - M_3 + M_5 \\
C_{1,2} &= M_3 + M_4 \\
C_{2,1} &= M_5 + M_6 \\
C_{2,2} &= M_0 - M_2 + M_4 - M_6
\end{aligned}
$$

Analisis Ejemplo 2.0

En este nuevo caso, para calcular el producto de matrices de $N \times N$, debemos calcular 7 productos de matrices de $\frac{n}{2} \times \frac{n}{2}$, seguido de 18 sumas (Las cuales tienen un costo $O(n^2)$).
Entonces, la ecuacion de recurrencia para el costo $T(n)$ es:

$$T(1) = c$$
$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

Cual es la solucion de dicha ecuacion?

Analisis Ejemplo 2.0

En este nuevo caso, para calcular el producto de matrices de $N \times N$, debemos calcular 7 productos de matrices de $\frac{n}{2} \times \frac{n}{2}$, seguido de 18 sumas (Las cuales tienen un costo $O(n^2)$).
Entonces, la ecuacion de recurrencia para el costo $T(n)$ es:

$$T(1) = c$$
$$T(n) = 7T\left(\frac{n}{2}\right) + n^2$$

Cual es la solucion de dicha ecuacion?
$T(n) \in O(n^{\log_2 7}) = O(n^{2.807})$ Notemos que esta solucion no es trivial, y es una pequeña mejora. Veamos como resolver ecuaciones de este tipo de forma mas trivial.

Paradigma “Divide y Venceras”

Teorema Maestro

Programacion Dinamica

Teorema Maestro

Dada una ecuación de recurrencia de la forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1, b \geq 2$$

Entonces, el teorema considera 3 casos. Primero se define.

Exponente Critico (E)

E se define como:

$$E = \log_b a = \frac{\log a}{\log b}$$

Caso 1

Si se cumple que:

$$f(n) \in O(n^{E-\epsilon}), \qquad \epsilon > 0$$

Entonces:

$$T(n) \in \Theta(n^E)$$

Ejemplo

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

Podemos extraer de la recurrencia los valores:

Ejemplo

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 8, b = 2, \quad f(n) = 1000n^2, \quad E = \log_2 8 = 3$$

Ejemplo

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 8, b = 2, \quad f(n) = 1000n^2, \quad E = \log_2 8 = 3$$

Entonces, chequeamos la condicion:

Ejemplo

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 8, b = 2, \quad f(n) = 1000n^2, \quad E = \log_2 8 = 3$$

Entonces, chequeamos la condicion:

$$f(n) \in O(n^{E-\epsilon}), \qquad \epsilon > 0$$

Ejemplo

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 8, b = 2, \quad f(n) = 1000n^2, \quad E = \log_2 8 = 3$$

Entonces, chequeamos la condicion:

$$f(n) \in O(n^{E-\epsilon}), \qquad \epsilon > 0$$

$$1000n^2 \in O(n^{3-\epsilon}), \qquad \epsilon > 0$$

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$1000n^2 \in O(n^2)$$

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$1000n^2 \in O(n^2)$$

Lo cual es cierto, por ende, la solucion a la recurrencia:

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$1000n^2 \in O(n^2)$$

Lo cual es cierto, por ende, la solucion a la recurrencia:

$$T(n) \in \Theta(n^3)$$

Caso 2

Si se cumple que:

$$f(n) \in \Theta(n^E \log^k n), \qquad k \geq 0$$

Entonces:

$$T(n) \in \Theta(n^E \log^{k+1} n)$$

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + 10n$$

Podemos extraer de la recurrencia los valores:

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + 10n$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = 10n, \quad E = \log_2 2 = 1$$

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + 10n$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = 10n, \quad E = \log_2 2 = 1$$

Entonces, chequeamos la condicion:

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + 10n$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = 10n, \quad E = \log_2 2 = 1$$

Entonces, chequeamos la condicion:

$$f(n) \in \Theta(n^E \log^k n), \qquad k \geq 0$$

Ejemplo

Dado que $k = 0$ y $E = 1$, entonces:

Ejemplo

Dado que $k = 0$ y $E = 1$, entonces:

$$10n \in \Theta(n)$$

Ejemplo

Dado que $k = 0$ y $E = 1$, entonces:

$$10n \in \Theta(n)$$

Lo cual es cierto, por ende:

Ejemplo

Dado que $k = 0$ y $E = 1$, entonces:

$$10n \in \Theta(n)$$

Lo cual es cierto, por ende:

$$T(n) \in \Theta(n \log n)$$

Caso 3

Si se cumple que:

$$f(n) \in \Omega(n^{E+\epsilon}), \qquad \epsilon > 0$$

Y adicionalmente se cumple:

$$af\left(\frac{n}{b}\right) \leq cf(n)$$

Para alguna constante $c < 1$ y n suficientemente largo. Entonces:

$$T(n) \in \Theta(f(n))$$

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Podemos extraer de la recurrencia los valores:

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = n^2, \quad E = \log_2 2 = 1$$

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = n^2, \quad E = \log_2 2 = 1$$

Entonces, chequeamos la condicion:

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = n^2, \quad E = \log_2 2 = 1$$

Entonces, chequeamos la condicion:

$$f(n) \in \Omega(n^{E+\epsilon}), \qquad \epsilon > 0$$

Ejemplo

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

Podemos extraer de la recurrencia los valores:

$$a = 2, b = 2, \quad f(n) = n^2, \quad E = \log_2 2 = 1$$

Entonces, chequeamos la condicion:

$$f(n) \in \Omega(n^{E+\epsilon}), \qquad \epsilon > 0$$

$$n^2 \in \Omega(n^{1+\epsilon}), \qquad \epsilon > 0$$

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$n^2 \in \Omega(n^2)$$

Lo cual es cierto, ahora chequeamos la segunda condicion:

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$n^2 \in \Omega(n^2)$$

Lo cual es cierto, ahora chequeamos la segunda condicion:

$$af\left(\frac{n}{b}\right) \leq cf(n), \qquad c < 1$$

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$n^2 \in \Omega(n^2)$$

Lo cual es cierto, ahora chequeamos la segunda condicion:

$$af\left(\frac{n}{b}\right) \leq cf(n), \qquad c < 1$$

$$2\left(\frac{n}{2}\right)^2 \leq cn^2 \rightarrow \frac{n^2}{2} \leq cn^2$$

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$n^2 \in \Omega(n^2)$$

Lo cual es cierto, ahora chequeamos la segunda condicion:

$$af\left(\frac{n}{b}\right) \leq cf(n), \qquad c < 1$$

$$2\left(\frac{n}{2}\right)^2 \leq cn^2 \rightarrow \frac{n^2}{2} \leq cn^2$$

Si elegimos $c = \frac{1}{2}$, entonces la condicion se cumple $\forall n$, y por ende:

Ejemplo

Si elegimos $\epsilon = 1$, entonces:

$$n^2 \in \Omega(n^2)$$

Lo cual es cierto, ahora chequeamos la segunda condicion:

$$af\left(\frac{n}{b}\right) \leq cf(n), \qquad c < 1$$

$$2\left(\frac{n}{2}\right)^2 \leq cn^2 \rightarrow \frac{n^2}{2} \leq cn^2$$

Si elegimos $c = \frac{1}{2}$, entonces la condicion se cumple $\forall n$, y por ende:

$$T(n) \in \Theta(n^2)$$

Notas sobre el Teorema Maestro

- Es posible que ninguno de los 3 casos aplique. En ese caso, el teorema falla en dar una solucion a la ecuacion de recurrencia.
- $f(n)$ debe ser positiva.
- $f(n)$ debe ser creciente.
- Se debe notar que el Teorema Maestro no resuelve recurrencias, solo da el orden asintotico de la solucion de la recurrencia.

Teorema Maestro Simplificado

Dada una ecuación de recurrencia de la forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1, b \geq 2$$

$$T(1) = c, \qquad c > 0$$

Si $f(n) \in \Theta(n^p)$ con $p \geq 0$, entonces:

$$T(n) \in \begin{cases} \Theta(n^E) & \text{si } p < E \\ \Theta(f(n)\log n) & \text{si } p == E \\ \Theta(f(n)) & \text{si } p > E \end{cases}$$

Ejercicios

$$T(n) = 2T(n/2) + n$$

Ejercicios

$$T(n) = 2T(n/2) + n \rightarrow T(n) \in \Theta(n \log n)$$

Ejercicios

$$T(n) = 2T(n/2) + n \rightarrow T(n) \in \Theta(n \log n)$$

$$T(n) = 2T(n/2) + 1$$

Ejercicios

$$T(n) = 2T(n/2) + n \rightarrow T(n) \in \Theta(n \log n)$$

$$T(n) = 2T(n/2) + 1 \rightarrow T(n) \in \Theta(n)$$

$$T(n) = 8T(n/2) + n^2$$

Ejercicios

$$T(n) = 2T(n/2) + n \rightarrow T(n) \in \Theta(n \log n)$$

$$T(n) = 2T(n/2) + 1 \rightarrow T(n) \in \Theta(n)$$

$$T(n) = 8T(n/2) + n^2 \rightarrow T(n) \in \Theta(n^3)$$

Paradigma “Divide y Venceras”

Teorema Maestro

Programacion Dinamica

Los que no recuerdan el pasado estan condenados a repetirlo.

Sindrome de Fibonacci

Recordemos la serie de Fibonacci:

$$f(0) = 0, \quad f(1) = 1, \quad f(n) = f(n-1) + f(n-2)$$

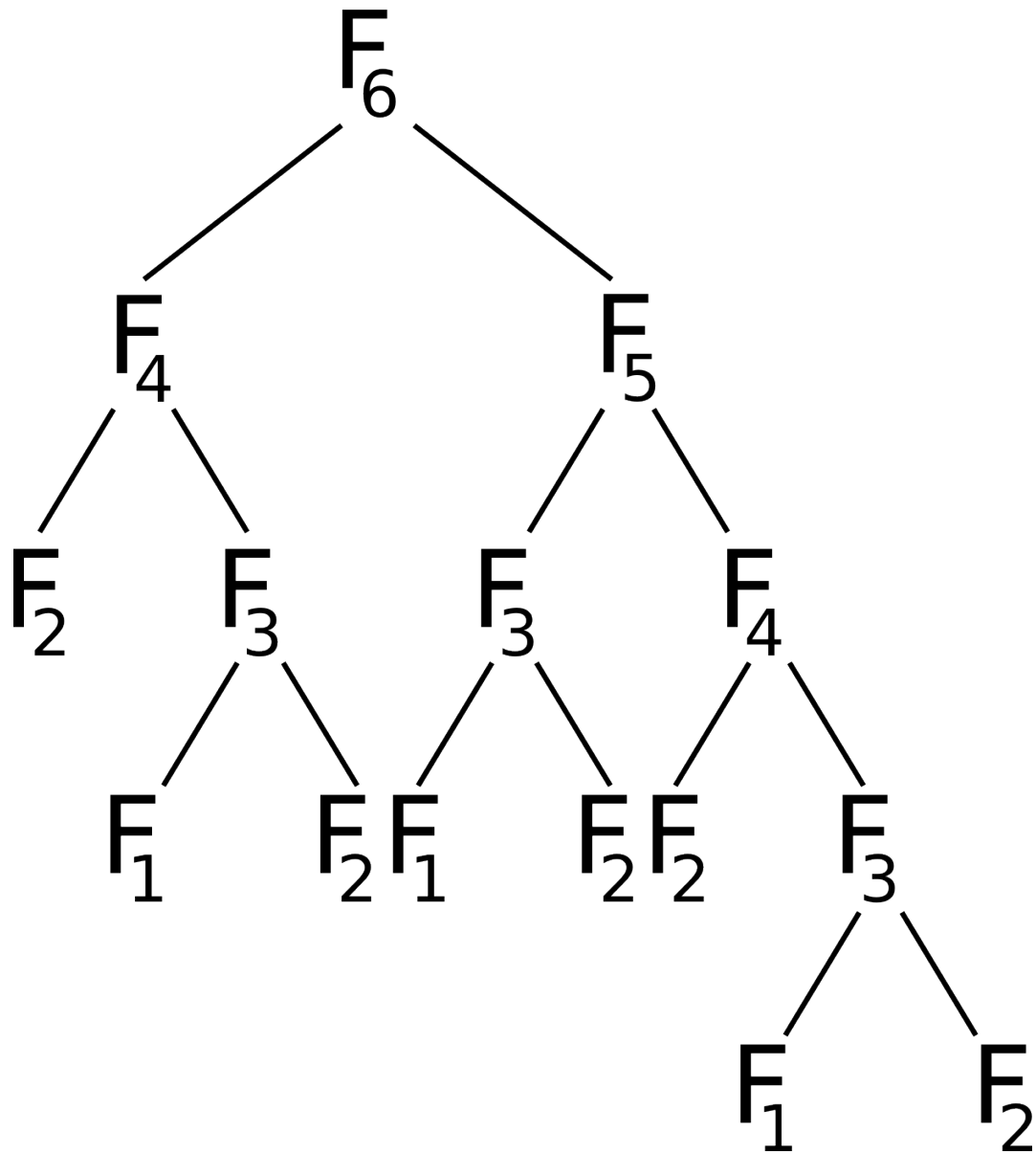
Si calculamos la serie de forma recursiva, usando la recurrencia:

$$f(0) = 0$$
$$f(1) = 1$$
$$f(2) = f(0) + f(1) = 1$$
$$f(3) = f(2) + f(1) = f(0) + f(1) + f(1) = 2$$
$$f(4) = f(3) + f(2) = f(0) + f(1) + f(1) + f(0) + f(1) = 3$$

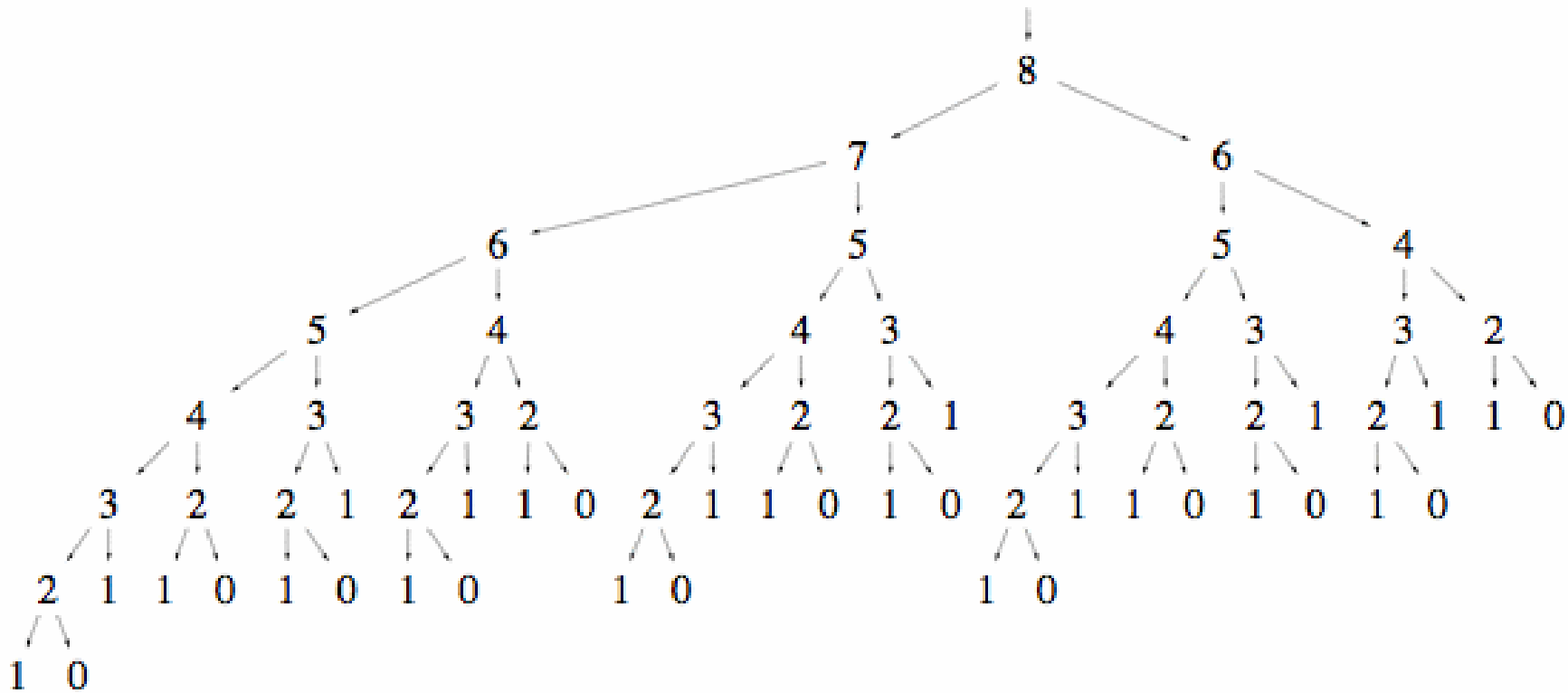
Sindrome de Fibonacci

Se puede ver claramente que al calcular Fibonacci en forma recursiva, se recalculan varias veces los diferentes valores ($f(2), f(3), f(4)$, etc).
Calcular el enesimo termino de la Serie de Fibonacci toma $O(e^n)$ llamadas recursivas.

Sindrome de Fibonacci



Sindrome de Fibonacci



Sindrome de Fibonacci

Entonces la gran pregunta es, es posible evitar ejecutar mas de una vez el calculo de cada numero de fibonacci? Es posible reusar el calculo del enesimo numero de Fibonacci para calcular los numeros de Fibonacci siguientes?

Reparacion de Algoritmos

Reparar un algoritmo que sufre del Sindrome de Fibonacci consiste en:

- Identificar los casos recursivos en los que se produce el sindrome.
- Usar una estructura de datos adecuada para almacenar (memorizar) los casos repetidos.
- Modificar la funcion recursiva tal que use y retorne el almacen de casos repetidos cada vez que se invoca con un caso previamente calculado.

Reparando Fibonacci

Consideremos el siguiente algoritmo:

```
unsigned long fibonacci(int n)
{
    long *p = new long[n + 1];
    p[0] = 0;
    p[1] = 1;
    long ret;

    for(int i = 2; i <= n; i++) {
        p[i] = p[i - 1] + p[i - 2];
    }

    ret = p[n];
    delete [] p;
    return ret;
}
```

Reparando Fibonacci

El algoritmo anterior permite calcular en enesimo numero de fibonacci en $O(n)$:

- Pero hace uso de $O(n)$ espacio.
- Es bastante obvio que el algoritmo no calcula ningun numero de Fibonacci del 2 en adelante mas de una vez (dentro de una llamada a la funcion).
- Lo que se hizo, es convertir un algoritmo $O(e^n)$ en uno $O(n)$.

Reparacion de Algoritmos

Entonces, una tecnica general para "reparar" algoritmos usando programacion dinamica es:

- Hacer un trueque entre el tiempo de ejecucion de un algoritmo y el espacio en memoria que usa.
- Este trueque es tipico y consiste en (en General):
  - Es posible que un algoritmo tome menos tiempo de ejecucion, al costo de usar mas espacio en memoria.
  - Es posible que un algoritmo use menos espacio en memoria, al costo de un mayor tiempo de ejecucion.
- Por ende, se debe experimentar haciendo el trueque, a lo cual llamamos reparar un algoritmo.
- El objetivo es reducir la complejidad computacional del algoritmo.

Ejemplo: Coeficientes Binomiales

Los coeficientes Binominales se calculan con la siguiente ecuacion de recurrencia:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{0} = \binom{n}{n} = 1, \quad \binom{0}{k} = 0$$

Construya el algoritmo recursivo que permite calcular $\binom{n}{k}$, y el algoritmo usando Programacion Dinamica que calcule lo mismo. Calcule la complejidad de ambos algoritmos.

Ejemplo: Coeficientes Binomiales

```
int binomial(int n, int k)
{
    if(k == 0) {
        return 1;
    } else if (n == 0) {
        return 0;
    } else {
        return binomial(n - 1, k - 1) +
               binomial(n - 1, k);
    }
}
```

Ejemplo: Coeficientes Binomiales

```
int binomial(int n, int k)
{
    int[][] A = new int[n+1][k+1];

    for(int i = 0; i < n; i++) {
        for(int j = 0; j < k; j++) {
            if(i == 0) {
                A[i][j] = 0;
            } else if(j == 0 || j == i) {
                A[i][j] = 1;
            } else {
                A[i][j] = A[i - 1][j - 1] + A[i - 1][j];
            }
        }
    }
    return A[n][k];
}
```

Ejemplo: Corte de Barras

Supongamos que tenemos barras de acero de largo L que deseamos cortar. La ganancia al vender las barras es una funcion del largo, y por ende, el dueño de la fabrica de barras desea maximizar la ganancia obtenida por cada barra.

Largo	1	2	3	4	5	6	7	8	9	10
Ganancia	1	5	8	9	10	17	17	20	24	30

Ejemplo: Corte de Barras

Deseamos calcular la maxima ganancia con una barra de largo L. Podemos relacionar recursivamente la ganancia maxima con un largo L con la ganancia maxima con los largos menores a L:

$$ganancia(L) = \max_{0 \leq i \leq L} \{p_i + ganancia(L - i)\}$$

Donde p_i es la ganancia con una barra de largo i.

Ejemplo: Corte de Barras

Lo cual se puede implementar de la siguiente forma:

```
int ganancia(int L)
{
    if(L == 0) return 0;

    g = -infinito;

    for(int i = 0; i < L; i++) {
        g = max(g, p[i] + ganancia(L - i));
    }

    return g;
}
```

Complejidad: Corte de Barras

Cual es la complejidad de este algoritmo? Es obvio que:

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

Es posible demostrar que:

$$T(n) = 2^n$$

Ya que:

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

El algoritmo intenta con todas las posibilidades de corte, y se queda con la de mayor ganancia. Es posible mejorar este algoritmo?

Programacion Dinamica: Corte de Barras

```
int cortar(int *p, int L)
{
    int *m = new int[L + 1];
    for(int i = 0; i <= L; i++) {
        m[i] = INT_MIN;
    }

    m[0] = 0;

    return cortarConMemoria(p, L, m);
}
```

Programacion Dinamica: Corte de Barras

```
int cortarConMemoria(int *p, int L, int *m)
{
    if(m[L] >= 0) {
        return m[L];
    }

    g = INT_MIN;

    for(int i = 1; i <= L; i++) {
        g = max(g, p[i] + cortarConMemoria(p, L - i, m));
    }

    m[L] = g;

    return g;
}
```

Complejidad: Corte de Barras

Cual es la complejidad temoporal del algoritmo mejorado con programacion dinamica?

- Dado que usando PD se resuelve cada sub-prooblema una sola vez, la complejidad no es la misma.
- El ciclo for dentro de cortarConMemoria se ejecuta L veces, y como dentro de dicho ciclo se resuelven los L sub-problemas una vez, la complejidad es $O(L^2)$.
- Esto es una gran mejora, desde $O(2^L)$ a $O(L^2)$.

El algoritmo intenta con todas las posibilidades de corte, pero soluciona cada caso exactamente una vez, y reutiliza dicho resultado multiples veces.

Ejercicio: Numeros de Catalan

Los numeros de Catalan estan definidos como:

$$C_0 = 1$$

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \qquad n \geq 0$$

Esto es equivalente a:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} \qquad n \geq 1$$

Construya el algoritmo recursivo que permite calcular C_n, y el algoritmo usando Programacion Dinamica que calcule lo mismo. Calcule la complejidad de ambos algoritmos.

Ejercicio: Numeros de Catalan

```
int catalan(int n)
{
    int ret = 0;

    if(n == 0) {
        ret = 1;
    } else {
        for(int i = 0; i < n; i++) {
            ret += catalan(i) * catalan(n - 1 - i);
        }
    }

    return ret;
}
```

Mejoras

Es posible mejorar mas estos algoritmos? Si!

- Es posible usar un arreglo global, independiente de las llamadas a la funcion, de manera que almacene todos los calculos hechos, y reusarlos al llamar la funcion.
- Con esta modificacion, cada llamada a funcion tomaria aproximadamente $O(1)$, pero usaria $O(n)$ o $O(n^2)$ de espacio en memoria permanentemente.

Bibliografia

Para esta unidad, se recomienda leer:

1. Capitulos 3 y 10 del Libro “Algoritmos Computacionales: Introduccion al Analisis y Diseño” de Sara Baase y Allen Van Gelder.