Analisis y Diseño de Algoritmos

Juan Gutiérrez

May 21, 2022

Fibonacci

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

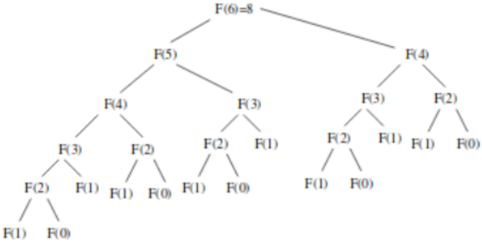
Fibonacci

Recibe: Un número n

Devuelve: $F(n)$

```
FIBONACCI-RECURSIVO(n)
if n == 0
    return 0
if n == 1
    return 1
return FIBONACCI-RECURSIVO(n − 1)+ FIBONACCI-RECURSIVO(n − 2)
```

Fibonacci


F(6)=8
F(5)
F(4)
F(4)
F(3)
F(3)
F(2)
F(3)
F(2)
F(2)
F(1)
F(2)
F(1)
F(1)
F(0)
F(2)
F(1)
F(1)
F(0)
F(1)
F(0)
F(1)
F(0)
F(1)
F(0)

Fibonacci

Recibe: Un número n

Devuelve: $F(n)$

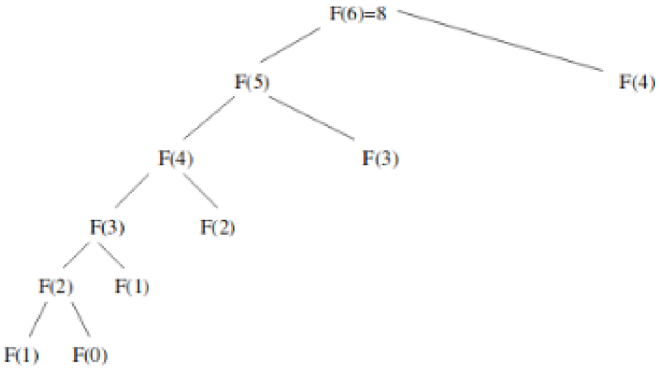
$\textsc{Fibonacci-Memoizado}(n)$

```
if M[n] ≠ −1
    return M[n]
else
    M[n] = FIBONACCI-MEMOIZADO(n−1)+FIBONACCI-MEMOIZADO(n−2)
    return M[n]
```

$\textsc{Main}(n)$

```
M[0] = 0
M[1] = 1
for i = 2 to n
    M[i] = −1
return FIBONACCI-MEMOIZADO(n)
```

Fibonacci



Fibonacci

Recibe: Un número n

Devuelve: $F(n)$

FIBONACCI-PROG-DIN(n)

$M[0] = 0$

$M[1] = 1$

for $i = 2$ **to** n

$\quad M[i] = M[i-1] + M[i-2]$

return $M[n]$

Coeficientes Binomiales

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

Coeficientes Binomiales

Recibe: Números naturales n, k con $n \geq k$.

Devuelve: $C(n, k)$

$\textsc{Coef-Bin-Prog-Din}(n, k)$

for $i = 0$ **to** n

    $C[i, i] = C[i, 0] = 1$

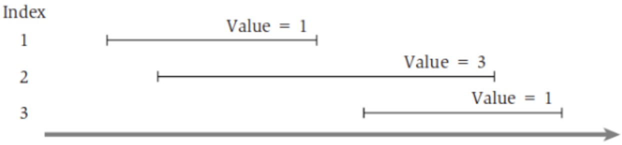
for $i = 1$ **to** n

    for $j = 1$ **to** $i - 1$

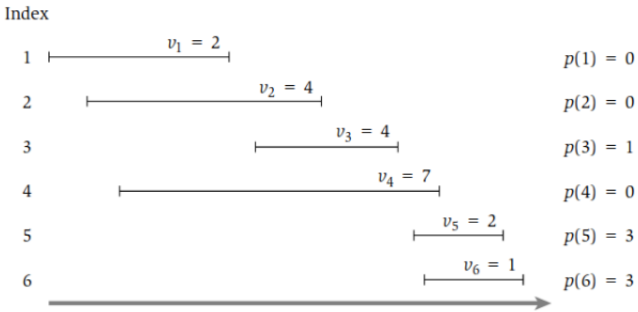
        $C[i, j] = C[i-1, j-1] + C[i-1, j]$

return $C[n, k]$

Intervalos disjuntos



Intervalos disjuntos



Intervalos disjuntos

$$OPT(j) = \max\{v_j + OPT(p(j)), OPT(j-1)\}.$$

Intervalos disjuntos

Recibe: Una secuencia de intervalos $\mathcal{I} = \{[s_i, f_i] : i = 1 \dots n\}$ con pesos v_i ordenados por la punta final. Un entero j.

Devuelve: El valor de un subconjunto de intervalos compatibles con peso máximo de entre los j primeros intervalos.

INTERVALOS-RECURSIVO$(\mathcal{I}, j)$

if $j = 0$

  return 0

$p(j) = \max\{i : [s_i, f_i] \cap [s_j, t_j] = \emptyset\}$

$\text{OPT}_1 =$ INTERVALOS-RECURSIVO$(\mathcal{I}, p(j))$

$\text{OPT}_2 =$ INTERVALOS-RECURSIVO$(\mathcal{I}, j-1)$

return $\max\{v_j + \text{OPT}_1, \text{OPT}_2\}$

Intervalos disjuntos

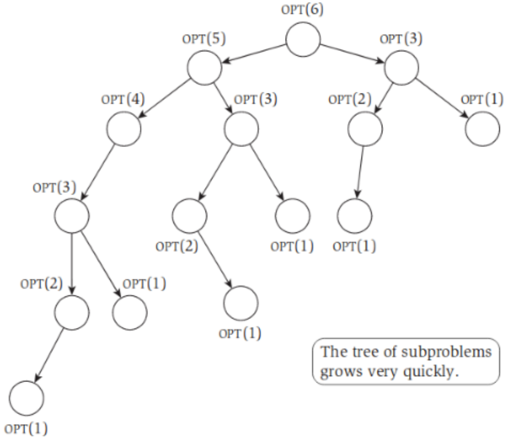


Figure 3: Tomada del libro Kleinberg, Algorithm Design

Intervalos disjuntos

Recibe: Una secuencia de intervalos $\mathcal{I} = [s_i, f_i] : i = 1 \ldots n$ con pesos v_i ordenados por la punta final. Un entero j.

Devuelve: El peso máximo de un subconjunto de intervalos con peso máximo de entre los j primeros intervalos.

INTERVALOS-MEMOIZADO$(\mathcal{I}, j)$

if $j = 0$
    return 0
if $M[j] \neq -1$
    return $M[j]$
$\text{OPT}_1 =$ INTERVALOS-MEMOIZADO$(\mathcal{I}, p(j))$
$\text{OPT}_2 =$ INTERVALOS-MEMOIZADO$(\mathcal{I}, j-1)$
$M[j] = \max\{v_j + \text{OPT}_1, \text{OPT}_2\}$
return $M[j]$

Intervalos disjuntos

Recibe: Una secuencia de intervalos $\mathcal{I} = [s_i, f_i] : i = 1 \ldots n$ con pesos v_i ordenados por la punta final. Un entero j. Un arreglo M resultante del algoritmo INTERVALOS-MEMOIZADO.

Devuelve: Un subconjunto de intervalos con peso máximo de entre los j primeros intervalos.

INTERVALOS-MEMOIZADO-SOLUCION($\mathcal{I}, j$)

if $j = 0$
    return $\emptyset$
if $v_j + M[p(j)] \geq M[j-1]$
    return $\{j\} \cup$ INTERVALOS-MEMOIZADO($\mathcal{I}, p(j)$)
else
    return INTERVALOS-MEMOIZADO($\mathcal{I}, j - 1$)

Intervalos disjuntos

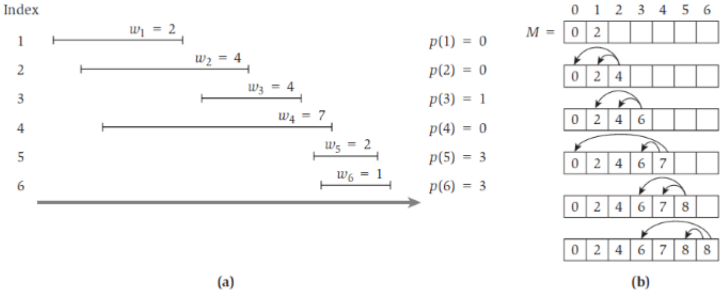
Recibe: Una secuencia de intervalos $\mathcal{I} = [s_i, f_i] : i = 1 \ldots n$ con pesos v_i ordenados por la punta final.

Devuelve: El valor de un subconjunto de intervalos compatibles con peso máximo de entre los j primeros intervalos.

INTERVALOS-PROG-DINAMICA$(\mathcal{I}, j)$

1: $M[0] = 0$
2: $p(j) = \max\{i : [s_i, f_i] \cap [s_j, t_j] = \emptyset\}$
3: **for** $j = 1$ **to** n
4: $\quad M[j] = \max\{v_j + M[p(j)], M[j-1]\}$
5: **return** $M[n]$

Intervalos disjuntos



Subsecuencia creciente máxima

Problema Max-Subsecuencia-Creciente. Dado un vector $A[1..n]$ encontrar una subsecuencia creciente de tamaño máximo en A.

Subsecuencia creciente máxima

$$OPT(i) = \max\{OPT(j) + 1 : j < i, A[j] < A[i]\}.$$

Subsecuencia creciente máxima

Recibe: Un arreglo $A[1..n]$.
Devuelve: El tamaño de una subsecuencia creciente máxima .

```
MAX-SUB-CREC-PROG-DINAMICA(A)
M[0] = 0
for i = 1 to n
    M[i] = max{M[j] : 0 ≤ j < i, A[j] < A[i]} + 1
return max{M[i] : 1 ≤ i ≤ n}
```

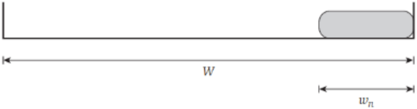
Subsecuencia creciente máxima

Sequence s_i	2	4	3	5	1	7	6	9	8
Length l_i	1	2	3	3	1	4	4	5	5
Predecessor p_i	–	1	1	2	–	4	4	6	6

Subset sum y mochila

Problema Max-Subset-Sum. Dado un conjunto $\{1, 2, \ldots, n\}$ de items cada uno con un peso natural w_i, y un número natural W, encontrar un subconjunto de items cuya suma de pesos es la mayor posible, pero menor o igual a W.

Subset sum y mochila



Subset sum y mochila

- Si $n \in X$ entonces OPT(n, W)=OPT$(n-1, W-w_n)+w_n$
- Si $n \notin X$ entonces OPT(n, W)=OPT$(n-1, W)$

Subset sum y mochila

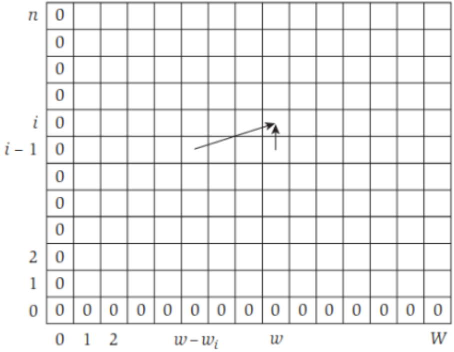
Recibe: Un arreglo $w[1..n]$ de números naturales (pesos) y un número natural W.

Devuelve: Una solución para el problema Subset-Sum.

SubsetSum-ProgDinamica(w, W)

for $j = 0$ **to** W
    $M[0, j] = 0$
for $i = 1$ **to** n
    for $j = 0$ **to** W
        if $w[i] > j$
            $M[i, j] = M[i-1, j]$
        else
            $M[i, j] = \max\{M[i-1, j], M[i-1, j-w[i]] + w[i]\}$
return $M[n][W]$

Subset sum y mochila



Subset sum y mochila

Knapsack size $W = 6$, items $w_1 = 2$, $w_2 = 2$, $w_3 = 3$

3							
2							
1							
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Initial values

3							
2							
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Filling in values for $i = 1$

3							
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Filling in values for $i = 2$

3	0	0	2	3	4	5	5
2	0	0	2	2	4	4	4
1	0	0	2	2	2	2	2
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

Filling in values for $i = 3$

Subset sum y mochila

- Si $n \in X$ entonces OPT(n, W)=OPT$(n-1, W-w_n)+v_n$
- Si $n \notin X$ entonces OPT(n, W)=OPT$(n-1, W)$

Min partition

Problema Min-Partition. Dado un arreglo A de números enteros no negativos y un número k, encontrar una partición con tamaño k de peso mínimo.

Min partition

$$OPT(n,k) = \begin{cases} \sum_{i=1}^{n} A[i] & \text{si } k = 1 \\ \min_{\ell=k}^{n} \max\{OPT(\ell-1, k-1), \sum_{j=\ell}^{n} A[j]\} & \text{en otro caso} \end{cases}$$

Min partition

$\textsc{MinPartition-ProgDinamica}(A, k)$

```
for i = 1 to n
    M[i, 1] = Σ_{p=1}^{n} A[p]
for i = 2 to n
    for j = 2 to k
        M[i, j] = ∞
        for ℓ = j to i
            cost = max{M[ℓ − 1][j − 1], Σ_{p=ℓ}^{n} A[p]}
            if cost < M[i, j]
                M[i, j] = cost
return M[n][k]
```

Gracias