Algoritmos y Estructuras de Datos II

Programación dinámica

30 de mayo de 2016

Clase de hoy

1. Repaso
   - Backtracking
2. Programación dinámica
   - Problema de la moneda
   - Problema de la mochila
3. Conclusión

Repaso

- cómo vs. qué
- 3 partes
  1. análisis de algoritmos
  2. tipos de datos
  3. técnicas de resolución de problemas
     - divide y vencerás
     - algoritmos voraces
     - backtracking
     - programación dinámica
     - recorrida de grafos

Backtracking

Problema de la moneda

- Sean $0 \leq i \leq n$ y $0 \leq j \leq k$,
- definimos $m(i,j) =$ "menor número de monedas necesarias para pagar exactamente el monto j con denominaciones $d_1, d_2, \ldots, d_i$."
- $$m(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = 0 \\ m(i-1,j) & d_i > j > 0 \wedge i > 0 \\ \min(m(i-1,j), 1 + m(i,j-d_i)) & j \geq d_i > 0 \wedge i > 0 \end{cases}$$
- En el peor caso es exponencial.

Backtracking

Problema de la mochila

- Sean $0 \leq i \leq n$ y $0 \leq j \leq W$,
- definimos $m(i,j) =$ “mayor valor alcanzable sin exceder la capacidad j con objetos $1, 2, \ldots, i$.”
- $$m(i,j) = \begin{cases} 0 & j = 0 \\ 0 & j > 0 \wedge i = 0 \\ m(i-1,j) & w_i > j > 0 \wedge i > 0 \\ \max(m(i-1,j), v_i + m(i-1,j-w_i)) & j \geq w_i > 0 \wedge i > 0 \end{cases}$$
- En el peor caso es exponencial.

Backtracking

Problema de los caminos de costo mínimo entre cada par de vértices

- Sean $1 \leq i, j \leq n$ y $0 \leq k \leq n$,
- definimos $m_k(i, j) =$ "menor costo posible para caminos de i a j cuyos vértices intermedios se encuentran en el conjunto $\{1,\ldots,k\}$."
- $$m_k(i,j) = \begin{cases} L[i,j] & k = 0 \\ \min(m_{k-1}(i,j), m_{k-1}(i,k) + m_{k-1}(k,j)) & k \geq 1 \end{cases}$$
- En el peor caso es exponencial.

Programación dinámica

- Método para transformar una definición recursiva en iterativa
- a través de la confección de una **tabla de valores**.
- Objetivo: evitar la reiteración de cómputos.
- Ejemplo: definición recursiva de la secuencia de Fibonacci.

Secuencia de Fibonacci

- $$f_n = \begin{cases} 1 & n \leq 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$
- Ya vimos que esta función recursiva es exponencial.
- La razón, el cálculo de f_n lleva a calcular
  - 2 veces f_{n-2},
  - 3 veces f_{n-3},
  - 5 veces f_{n-4},
  - etc.

¿Cómo podemos evitar tantos recálculos?

- Llevando una tabla de valores calculados.
- Comenzando desde los casos bases.
- Sea f un arreglo de 0 a n.
  - f[0]:= 1
  - f[1]:= 1
  - f[2]:= f[1]+f[0]
  - f[3]:= f[2]+f[1]
  - etc

Fibonacci a través de una tabla

```
fun fib(n: nat) ret r: nat
    var f: array[0..max(n,1)] of nat
    f[0]:= 1
    f[1]:= 1
    for i:= 2 to n do f[i]:= f[i-1] + f[i-2] od
    r:= f[n]
end fun
```

¡Este algoritmo es lineal!

Problema de la moneda
Backtracking

Vimos la definición

$$m(i,j) = \begin{cases} 0 & j = 0 \\ \infty & j > 0 \wedge i = 0 \\ m(i-1,j) & d_i > j > 0 \wedge i > 0 \\ \min(m(i-1,j), 1 + m(i, j-d_i)) & j \geq d_i > 0 \wedge i > 0 \end{cases}$$

que puede ser exponencial debido a que tiene dos llamadas recursivas en el último caso.

Problema de la moneda
Confección de una tabla

- Habiendo dos parámetros, la tabla será una matriz en vez de un vector como en el caso de Fibonacci.
- Los casos base corresponden al llenado de la primera columna y primera fila de la matriz.
- Como todas las llamadas recursivas se realizan decrementando el "parámetro *i*" o manteniendolo igual pero en ese caso decrementando el "parámetro *j*", se propone el siguiente método de llenado de la matriz:
  - fila por fila, desde la primera a la última, de modo de que el valor correspondiente a $m(i-1, j)$ ya esté computado al calcular el valor correspondiente a $m(i, j)$
  - dentro de cada fila, desde la primer columna hasta la última, de modo de que el valor correspondiente a $m(i, j - d_i)$ ya esté computado al calcular $m(i, j)$

Problema de la moneda
Programación dinámica

```
fun cambio(d:array[1..n] of nat, k: nat) ret r: nat
    var m: array[0..n,0..k] of nat
    for i:= 0 to n do m[i,0]:= 0 od
    for j:= 1 to k do m[0,j]:= ∞ od
    for i:= 1 to n do
        for j:= 1 to k do
            if d[i] > j then m[i,j]:= m[i-1,j]
            else m[i,j]:= min(m[i-1,j],1+m[i,j-d[i]])
            fi
        od
    od
    r:= m[n,k]
end fun
```

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0																	
1																	
2																	
3																	

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0																
1	0																
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0																
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞													
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞													
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1												
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞											
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞										
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞									
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2								
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0																
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞															
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞															
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1														
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0																

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0	∞	1	∞	1	∞	2										

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0	∞	1	∞	1	∞	2	1									

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0	∞	1	∞	1	∞	2	1	2								

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0	∞	1	∞	1	∞	2	1	2	2							

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0	∞	1	∞	1	∞	2	1	2	2	3	2	3	3			

Problema de la moneda

Ejemplo con denominaciones $d_1 = 4$, $d_2 = 2$ y $d_3 = 7$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	1	∞	∞	∞	2	∞	∞	∞	3	∞	∞	∞	4
2	0	∞	1	∞	1	∞	2	∞	2	∞	3	∞	3	∞	4	∞	4
3	0	∞	1	∞	1	∞	2	1	2	2	3	2	3	3	2	3	3

Problema de la mochila

Backtracking

Vimos la definición

$$m(i,j) = \begin{cases} 0 & j = 0 \\ 0 & j > 0 \wedge i = 0 \\ m(i-1,j) & w_i > j > 0 \wedge i > 0 \\ \max(m(i-1,j), v_i + m(i-1,j-w_i)) & j \geq w_i > 0 \wedge i > 0 \end{cases}$$

que puede ser exponencial debido a que tiene dos llamadas recursivas en el último caso.

Problema de la mochila
Confección de una tabla

- Habiendo dos parámetros, la tabla será nuevamente una matriz.
- Los casos base corresponden al llenado de la primera columna y primera fila de la matriz.
- Como todas las llamadas recursivas se realizan decrementando el "parámetro *i*", la única condición necesaria para el llenado de la tabla es proceder fila por fila, no importa el orden de llenado dentro de cada fila.

Problema de la mochila

Programación dinámica

```
fun mochila(v:array[1..n] of valor, w:array[1..n] of nat, W: nat)
                                                      ret r: valor
    var m: array[0..n,0..W] of valor
    for i:= 0 to n do m[i,0]:= 0 od
    for j:= 1 to W do m[0,j]:= 0 od
    for i:= 1 to n do
        for j:= 1 to W do
            if w[i] > j then m[i,j]:= m[i-1,j]
            else m[i,j]:= max(m[i-1,j],v[i]+m[i-1,j-w[i]])
            fi
        od
    od
    r:= m[n,W]
end fun
```

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0																	
1																	
2																	
3																	
4																	

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0																
2	0																
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0									
2	0																
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3								
2	0																
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3							
2	0																
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0																
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0												
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2									
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3								
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3				
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5			
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0																
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2										
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3									
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3					
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5		
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	6
4	0																

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	6
4	0	0	0														

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	6
4	0	0	0	2	2	2	2										

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	6
4	0	0	0	2	2	2	2	3	4	4							

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	6
4	0	0	0	2	2	2	2	3	4	4	5	5	5	5	5		

Problema de la mochila

Ejemplo con valores $v_1 = 3$, $v_2 = 2$, $v_3 = 3$, $v_4 = 2$, $w_1 = 8$, $w_2 = 5$, $w_3 = 7$ y $w_4 = 3$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3
2	0	0	0	0	0	2	2	2	3	3	3	3	3	5	5	5	5
3	0	0	0	0	0	2	2	3	3	3	3	3	5	5	5	6	6
4	0	0	0	2	2	2	2	3	4	4	5	5	5	5	5	7	7

Conclusión

- Algoritmos voraces
  - Cuando tenemos un criterio de selección que garantiza optimalidad
- Backtracking
  - Cuando no tenemos un criterio así
  - solución top-down
  - en general es exponencial
- Programación dinámica
  - construye una tabla bottom-up
  - evita repetir cálculos
  - pero realiza algunos cálculos inútiles.