

Algoritmos y Estructuras de Datos II

Programación dinámica

5 y 7 de junio de 2017

Clase de hoy

- 1 Backtracking
- 2 Programación dinámica
 - Problema de la moneda
 - Problema de la mochila
- 3 Algoritmo de Floyd
 - Problema del camino de costo mínimo
 - Algoritmo de Floyd
 - Ejemplo
 - Reconstrucción del camino
 - Otras reconstrucciones
- 4 Conclusión

Backtracking

Problema de la moneda

- Sean $0 \leq i \leq n$ y $0 \leq j \leq k$,
- definimos $cambio(i, j) =$ “menor número de monedas necesarias para pagar exactamente el monto j con denominaciones d_1, d_2, \dots, d_i .”



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

- En el peor caso es exponencial.

Backtracking

Problema de la mochila

- Sean $0 \leq i \leq n$ y $0 \leq j \leq W$,
- definimos $mochila(i, j) =$ “mayor valor alcanzable sin exceder la capacidad j con objetos $1, 2, \dots, i$.”
-

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

- 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 $camino_k(i, j) =$ “menor costo posible para caminos de i a j cuyos vértices intermedios se encuentran en el conjunto $\{1, \dots, k\}$.”



$$camino_k(i, j) = \begin{cases} L[i, j] & k = 0 \\ \min(camino_{k-1}(i, j), camino_{k-1}(i, k) + camino_{k-1}(k, j)) & k \geq 1 \end{cases}$$

- Es exponencial (3^n).

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 cálculos.
- 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

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

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 $cambio(i - 1, j)$ ya esté computado al calcular el valor correspondiente a $cambio(i, j)$
 - dentro de cada fila, desde la primer columna hasta la última, de modo de que el valor correspondiente a $cambio(i, j - d_i)$ ya esté computado al calcular $cambio(i, j)$

Problema de la moneda

Programación dinámica

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

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

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 moch: array[0..n,0..W] of valor
    for i:= 0 to n do moch[i,0]:= 0 od
    for j:= 1 to W do moch[0,j]:= 0 od
    for i:= 1 to n do
        for j:= 1 to W do
            if w[i] > j then moch[i,j]:= moch[i-1,j]
            else moch[i,j]:= max(moch[i-1,j],v[i]+moch[i-1,j-w[i]])
            fi
        od
    od
    r:= moch[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

Problema del camino de costo mínimo entre todo par de vértices

Backtracking

Vimos la definición

$$camino_k(i, j) = \begin{cases} L[i, j] & k = 0 \\ \min(camino_{k-1}(i, j), camino_{k-1}(i, k) + camino_{k-1}(k, j)) & k \geq 1 \end{cases}$$

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

Problema del camino de costo mínimo

Confección de una tabla

- Habiendo tres parámetros, la tabla será un arreglo tridimensional.
- El caso base corresponde al llenado de la matriz $m[0, i, j]$.
- Como todas las llamadas recursivas se realizan decrementando el “parámetro k ”, la única condición necesaria para el llenado de la tabla es proceder desde k igual a 0 hasta k igual a n .
 - Primero se copia $m[0, i, j] := L[i, j]$ para todo i, j .
 - Luego, para todo $k > 0$, y para todo i, j se asigna $m[k, i, j] := \min(m[k - 1, i, j], m[k - 1, i, k] + m[k - 1, k, j])$

Problema del camino de costo mínimo

Observaciones interesantes

- Dijimos “Luego, para todo $k > 0$, y para todo i, j se asigna $m[k, i, j] := \min(m[k - 1, i, j], m[k - 1, i, k] + m[k - 1, k, j])$ ”
- Observar qué pasa al calcular la fila k de la matriz k
 - $m[k, k, j] := \min(m[k - 1, k, j], m[k - 1, k, k] + m[k - 1, k, j])$,
 - o sea, $m[k, k, j] := m[k - 1, k, j]$, ¡no cambia!
- Observar qué pasa al calcular la columna k de la matriz k
 - $m[k, i, k] := \min(m[k - 1, i, k], m[k - 1, i, k] + m[k - 1, k, k])$,
 - o sea, $m[k, i, k] := m[k - 1, i, k]$, ¡tampoco cambia!
- Para calcular la celda i, j de la matriz k , sólo se necesitan:
 - la misma celda de la matriz anterior
 - la celda i, k (que está en la columna k) de la matriz anterior
 - la celda k, j (que está en la fila k) de la matriz anterior
- ¡Entonces podemos hacer todo en una única matriz!

Problema del camino de costo mínimo

Programación dinámica

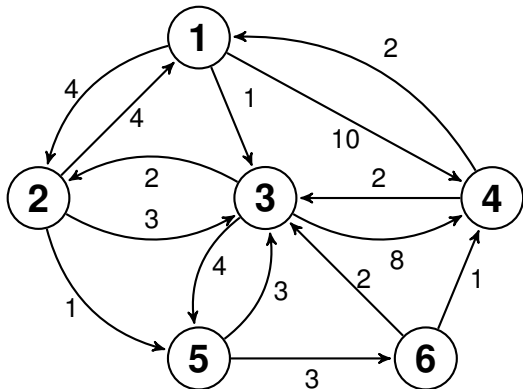
```
fun Floyd(L:array[1..n,1..n] of costo) ret m: array[1..n,1..n] of costo
  copiar L a m
  for k:= 1 to n do
    for i:= 1 to n do
      for j:= 1 to n do
        m[i,j]:= min(m[i,j],m[i,k]+m[k,j])
      od
    od
  od
end fun
```

Problema del camino de costo mínimo

Programación dinámica

```
fun Floyd(L:array[1..n,1..n] of costo) ret m: array[1..n,1..n] of costo
  for i:= 1 to n do
    for j:= 1 to n do
      m[i,j]:= L[i,j]
    od
  od
  for k:= 1 to n do
    for i:= 1 to n do
      for j:= 1 to n do
        m[i,j]:= min(m[i,j],m[i,k]+m[k,j])
      od
    od
  od
end fun
```


Grafo



Matriz de adyacencia L

$m_0 =$

0	4	1	10	∞	∞
4	0	3	∞	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	∞	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	∞	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	∞	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	∞	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_1

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

 = m_1

Calculando m_2

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_2

0	4	1	10	∞	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_2

0	4	1	10	5	∞
4	0	3	14	1	∞
∞	2	0	8	4	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_2

0	4	1	10	5	∞
4	0	3	14	1	∞
6	2	0	8	3	∞
2	6	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_2

0	4	1	10	5	∞
4	0	3	14	1	∞
6	2	0	8	3	∞
2	6	2	0	7	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_2

0	4	1	10	5	∞
4	0	3	14	1	∞
6	2	0	8	3	∞
2	6	2	0	7	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

 = m_2

Calculando m_3

0	4	1	10	5	∞
4	0	3	14	1	∞
6	2	0	8	3	∞
2	6	2	0	7	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_3

0	4	1	10	5	∞
4	0	3	14	1	∞
6	2	0	8	3	∞
2	6	2	0	7	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_3

0	3	1	9	4	∞
4	0	3	14	1	∞
6	2	0	8	3	∞
2	6	2	0	7	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_3

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	6	2	0	7	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_3

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

Calculando m_3

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
∞	∞	2	1	∞	0

Calculando m_3

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

 = m_3

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
8	4	2	1	5	0

Calculando m_4

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
3	4	2	1	5	0

= m_4

Calculando m_5

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_5

0	3	1	9	4	∞
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_5

0	3	1	9	4	7
4	0	3	11	1	∞
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_5

0	3	1	9	4	7
4	0	3	11	1	4
6	2	0	8	3	∞
2	4	2	0	5	∞
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_5

0	3	1	9	4	7
4	0	3	11	1	4
6	2	0	8	3	6
2	4	2	0	5	∞
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_5

0	3	1	9	4	7
4	0	3	11	1	4
6	2	0	8	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_5

0	3	1	9	4	7
4	0	3	11	1	4
6	2	0	8	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

 = m_5

Calculando m_6

0	3	1	9	4	7
4	0	3	11	1	4
6	2	0	8	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_6

0	3	1	9	4	7
4	0	3	11	1	4
6	2	0	8	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_6

0	3	1	8	4	7
4	0	3	11	1	4
6	2	0	8	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_6

0	3	1	8	4	7
4	0	3	5	1	4
6	2	0	8	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_6

0	3	1	8	4	7
4	0	3	5	1	4
6	2	0	7	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_6

0	3	1	8	4	7
4	0	3	5	1	4
6	2	0	7	3	6
2	4	2	0	5	8
9	5	3	11	0	3
3	4	2	1	5	0

Calculando m_6

0	3	1	8	4	7
4	0	3	5	1	4
6	2	0	7	3	6
2	4	2	0	5	8
6	5	3	4	0	3
3	4	2	1	5	0

 = m_6

Reconstrucción del camino

```
fun Floyd(L:array[1..n,1..n] of costo) ret m: array[1..n,1..n] of costo  
                                ret E: array[1..n,1..n] of nat
```

copiar L a m

inicializar las celdas de E en 0

```
for k:= 1 to n do
```

```
    for i:= 1 to n do
```

```
        for j:= 1 to n do
```

```
            if m[i,k]+m[k,j] < m[i,j] then m[i,j]:= m[i,k]+m[k,j]  
                                           E[i,j]:= k
```

```
        fi
```

```
    od
```

```
od
```

```
od
```

```
end fun
```

Matriz de adyacencia L

m_0					
0	4	1	10	∞	∞
4	0	3	∞	1	∞
∞	2	0	8	4	∞
2	∞	2	0	∞	∞
∞	∞	3	∞	0	3
∞	∞	2	1	∞	0

E_0					
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Calculando m_1

m_1						E_1					
0	4	1	10	∞	∞	0	0	0	0	0	0
4	0	3	14	1	∞	0	0	0	1	0	0
∞	2	0	8	4	∞	0	0	0	0	0	0
2	6	2	0	∞	∞	0	1	0	0	0	0
∞	∞	3	∞	0	3	0	0	0	0	0	0
∞	∞	2	1	∞	0	0	0	0	0	0	0

Calculando m_2

m_2						E_2					
0	4	1	10	5	∞	0	0	0	0	2	0
4	0	3	14	1	∞	0	0	0	1	0	0
6	2	0	8	3	∞	2	0	0	0	2	0
2	6	2	0	7	∞	0	1	0	0	2	0
∞	∞	3	∞	0	3	0	0	0	0	0	0
∞	∞	2	1	∞	0	0	0	0	0	0	0

Calculando m_3

m_3						E_3					
0	3	1	9	4	∞	0	3	0	3	3	0
4	0	3	11	1	∞	0	0	0	3	0	0
6	2	0	8	3	∞	2	0	0	0	2	0
2	4	2	0	5	∞	0	3	0	0	3	0
9	5	3	11	0	3	3	3	0	3	0	0
8	4	2	1	5	0	3	3	0	0	3	0

Calculando m_4

m_4						E_4					
0	3	1	9	4	∞	0	3	0	3	3	0
4	0	3	11	1	∞	0	0	0	3	0	0
6	2	0	8	3	∞	2	0	0	0	2	0
2	4	2	0	5	∞	0	3	0	0	3	0
9	5	3	11	0	3	3	3	0	3	0	0
3	4	2	1	5	0	4	3	0	0	3	0

Calculando m_5

m_5						E_5					
0	3	1	9	4	7	0	3	0	3	3	5
4	0	3	11	1	4	0	0	0	3	0	5
6	2	0	8	3	6	2	0	0	0	2	5
2	4	2	0	5	8	0	3	0	0	3	5
9	5	3	11	0	3	3	3	0	3	0	0
3	4	2	1	5	0	4	3	0	0	3	0

Calculando m_6

m_6						E_6					
0	3	1	8	4	7	0	3	0	6	3	5
4	0	3	5	1	4	0	0	0	6	0	5
6	2	0	7	3	6	2	0	0	6	2	5
2	4	2	0	5	8	0	3	0	0	3	5
6	5	3	4	0	3	6	3	0	6	0	0
3	4	2	1	5	0	4	3	0	0	3	0

Problema de la moneda

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

Problema de la mochila

```
fun mochila(v:array[1..n] of valor, w:array[1..n] of nat, W: nat) ret nr: array[1..n] of bool
  var m: array[0..n,0..W] of valor
      r,s: nat
  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:= n
  s:= W
  while m[r,s] > 0 do
    if m[r,s] = m[r-1,s] then nr[r]:= false
    else nr[r]:= true
      s:= s-w[r]
    fi
    r:= r-1
  od
end fun
```

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.