Algoritmos y Estructuras de Datos II

Programación dinámica

27 de mayo de 2019

Clase de hoy

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 $d_1, d_2, \ldots, d_i$.”
-

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

- 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 $1, 2, \ldots, i$."
-

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

- 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 $\{$1,...,k$\}$."
-

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

- 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 cómputos.
- Ejemplo: definición recursiva de la secuencia de Fibonacci.

Secuencia de Fibonacci

- $$f_n = \begin{cases} n & n \leq 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$
- 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]:= 0
  - 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]:= 0
    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

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

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]:= ∞ **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 $cams[0, i, j]$.
- 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 $cams[0, i, j] := L[i, j]$ para todo i, j.
  - Luego, para todo $k > 0$, y para todo i, j se asigna $cams[k, i, j] := min(cams[k-1, i, j], cams[k-1, i, k] + cams[k-1, k, j])$

Problema del camino de costo mínimo

Programación dinámica

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

Problema del camino de costo mínimo

Programación dinámica

```
fun camino(L:array[1..n,1..n] of costo) ret cams: array[1..n,1..n] of co
   for i:= 1 to n do
      for j:= 1 to n do cams[0,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
            cams[k,i,j]:= min(cams[k-1,i,j],cams[k-1,i,k]+cams[k-1,k,j])
         od
      od
   od
end fun
```

Problema del camino de costo mínimo

Primera observación interesante

- Dijimos:
  "para todo $k > 0$, y para todo i, j se asigna $cams[k, i, j] := min(cams[k-1, i, j], cams[k-1, i, k] + cams[k-1, k, j])$"
- ¿Qué pasa al calcular la fila k de la matriz k-ésima?
  - $cams[k, k, j] := min(cams[k-1, k, j], cams[k-1, k, k] + cams[k-1, k, j])$,
  - o sea,
    $cams[k, k, j] := min(cams[k-1, k, j], 0 + cams[k-1, k, j])$,
  - o sea, $cams[k, k, j] := cams[k-1, k, j]$.
- ¡La fila k de la matriz k-ésima no cambia!

Problema del camino de costo mínimo

Segunda observación interesante

- ¿Qué pasa al calcular la columna k de la matriz k-ésima?
  - $cams[k, i, k] :=$ $min(cams[k-1, i, k], cams[k-1, i, k] + cams[k-1, k, k])$,
  - o sea, $cams[k, i, k] := min(cams[k-1, i, k], cams[k-1, i, k] + 0)$,
  - o sea, $m[k, i, k] := m[k-1, i, k]$.
- ¡La columna k de la matriz k-ésima tampoco cambia!

Problema del camino de costo mínimo

Tercera observación interesante

- Para calcular la celda i, j de la matriz k-ésima ($k > 0$), se calcula:
  $cams[k, i, j] :=$
  $min(cams[k-1, i, j], cams[k-1, i, k] + cams[k-1, k, j])$.
- en este cálculo sólo se necesitan:
  - la misma celda de la matriz anterior ($cams[k-1, i, j]$)
  - la celda i, k de la matriz anterior ($cams[k-1, i, k]$)
    esta celda está en la columna k, ¡no cambia!
  - la celda k, j de la matriz anterior ($cams[k-1, k, j]$)
    esta celda está en la fila k, ¡no cambia!
- ¡Entonces podemos hacer todo en una única matriz!

Problema del camino de costo mínimo

Programación dinámica

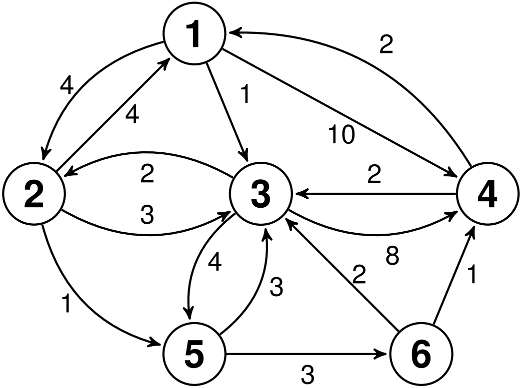
```
fun Floyd(L:array[1..n,1..n] of costo) ret cams: array[1..n,1..n] of cos
    copiar L a cams
    for k:= 1 to n do
        for i:= 1 to n do
            for j:= 1 to n do
                cams[i,j]:= min(cams[i,j],cams[i,k]+cams[k,j])
            od
        od
    od
end fun
```

Problema del camino de costo mínimo

Programación dinámica

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

Grafo



Matriz de adyacencia L

$$cams_0 = \begin{array}{|c|c|c|c|c|c|} \hline 0 & 4 & 1 & 10 & \infty & \infty \\ \hline 4 & 0 & 3 & \infty & 1 & \infty \\ \hline \infty & 2 & 0 & 8 & 4 & \infty \\ \hline 2 & \infty & 2 & 0 & \infty & \infty \\ \hline \infty & \infty & 3 & \infty & 0 & 3 \\ \hline \infty & \infty & 2 & 1 & \infty & 0 \\ \hline \end{array}$$

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

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

Calculando $cams_1$

$$\begin{array}{|c|c|c|c|c|c|}
\hline
0 & 4 & 1 & 10 & \infty & \infty \\ \hline
4 & 0 & 3 & 14 & 1 & \infty \\ \hline
\infty & 2 & 0 & 8 & 4 & \infty \\ \hline
2 & 6 & 2 & 0 & \infty & \infty \\ \hline
\infty & \infty & 3 & \infty & 0 & 3 \\ \hline
\infty & \infty & 2 & 1 & \infty & 0 \\ \hline
\end{array} = cams_1$$

Calculando $cams_2$

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

Calculando $cams_2$

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

Calculando $cams_2$

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 $cams_2$

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 $cams_2$

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 $cams_2$

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 4 & 1 & 10 & 5 & \infty \\ \hline 4 & 0 & 3 & 14 & 1 & \infty \\ \hline 6 & 2 & 0 & 8 & 3 & \infty \\ \hline 2 & 6 & 2 & 0 & 7 & \infty \\ \hline \infty & \infty & 3 & \infty & 0 & 3 \\ \hline \infty & \infty & 2 & 1 & \infty & 0 \\ \hline \end{array} = cams_2$$

Calculando $cams_3$

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 $cams_3$

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 $cams_3$

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 $cams_3$

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 $cams_3$

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 $cams_3$

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 $cams_3$

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 3 & 1 & 9 & 4 & \infty \\ \hline 4 & 0 & 3 & 11 & 1 & \infty \\ \hline 6 & 2 & 0 & 8 & 3 & \infty \\ \hline 2 & 4 & 2 & 0 & 5 & \infty \\ \hline 9 & 5 & 3 & 11 & 0 & 3 \\ \hline 8 & 4 & 2 & 1 & 5 & 0 \\ \hline \end{array} = cams_3$$

Calculando $cams_4$

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 $cams_4$

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 $cams_4$

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 $cams_4$

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 $cams_4$

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 $cams_4$

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 $cams_4$

$$\begin{array}{|c|c|c|c|c|c|} \hline 0 & 3 & 1 & 9 & 4 & \infty \\ \hline 4 & 0 & 3 & 11 & 1 & \infty \\ \hline 6 & 2 & 0 & 8 & 3 & \infty \\ \hline 2 & 4 & 2 & 0 & 5 & \infty \\ \hline 9 & 5 & 3 & 11 & 0 & 3 \\ \hline 3 & 4 & 2 & 1 & 5 & 0 \\ \hline \end{array} = cams_4$$

Calculando $cams_5$

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 $cams_5$

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 $cams_5$

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 $cams_5$

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 $cams_5$

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 $cams_5$

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 $cams_5$

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 3 & 1 & 9 & 4 & 7 \\ \hline 4 & 0 & 3 & 11 & 1 & 4 \\ \hline 6 & 2 & 0 & 8 & 3 & 6 \\ \hline 2 & 4 & 2 & 0 & 5 & 8 \\ \hline 9 & 5 & 3 & 11 & 0 & 3 \\ \hline 3 & 4 & 2 & 1 & 5 & 0 \\ \hline \end{array} = cams_5$$

Calculando $cams_6$

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 $cams_6$

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 $cams_6$

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 $cams_6$

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 $cams_6$

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 $cams_6$

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 $cams_6$

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

$= cams_6$

Reconstrucción del camino

```
fun Floyd(L:array[1..n,1..n] of costo) ret cams: array[1..n,1..n] of cos
                                        ret E: array[1..n,1..n] of nat
    copiar L a cams
    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 cams[i,k]+cams[k,j] < cams[i,j]
                then cams[i,j]:= cams[i,k]+cams[k,j]
                     E[i,j]:= k
                fi
            od
        od
    od
```

Matriz de adyacencia L

$cams_0$

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 $cams_1$

$cams_1$

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

E_1

0	0	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Calculando $cams_2$

$cams_2$

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

E_2

0	0	0	0	2	0
0	0	0	1	0	0
2	0	0	0	2	0
0	1	0	0	2	0
0	0	0	0	0	0
0	0	0	0	0	0

Calculando $cams_3$

$cams_3$

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

E_3

0	3	0	3	3	0
0	0	0	3	0	0
2	0	0	0	2	0
0	3	0	0	3	0
3	3	0	3	0	0
3	3	0	0	3	0

Calculando $cams_4$

$cams_4$

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

E_4

0	3	0	3	3	0
0	0	0	3	0	0
2	0	0	0	2	0
0	3	0	0	3	0
3	3	0	3	0	0
4	3	0	0	3	0

Calculando $cams_5$

$cams_5$

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

E_5

0	3	0	3	3	5
0	0	0	3	0	5
2	0	0	0	2	5
0	3	0	0	3	5
3	3	0	3	0	0
4	3	0	0	3	0

Calculando $cams_6$

$cams_6$

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

E_6

0	3	0	6	3	5
0	0	0	6	0	5
2	0	0	6	2	5
0	3	0	0	3	5
6	3	0	6	0	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 cam: array[0..n,0..k] of nat
        r,s: nat
    for i:= 0 to n do cam[i,0]:= 0 od
    for j:= 1 to k do cam[0,j]:= ∞ 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
    for i:= 0 to n do nr[i]:= 0 od
    nr[0]:= cam[n,k]
    if cam[n,k] ≠ ∞ then
      r:= n
      s:= k
      while cam[r,s] > 0 do
          if cam[r,s] = cam[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 moch: array[0..n,0..W] of valor
        r,s: nat
    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:= n
    s:= W
    while moch[r,s] > 0 do
        if moch[r,s] = moch[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.