Tema 24: Técnicas de diseño ascendente de algoritmos

Informática (2013–14)

José A. Alonso Jiménez

Grupo de Lógica Computacional
Departamento de Ciencias de la Computación e I.A.
Universidad de Sevilla

Tema 24: Técnicas de diseño ascendente de algoritmos

1. Programación dinámica
2. Fibonacci como ejemplo de programación dinámica
3. Producto de cadenas de matrices (PCM)
4. Árboles binarios de búsqueda optimales (ABBO)
5. Caminos mínimos entre todos los pares de nodos de un grafo(CM)
6. Problema del viajante (PV)

Tema 24: Técnicas de diseño ascendente de algoritmos

Divide y vencerás vs programación dinámica

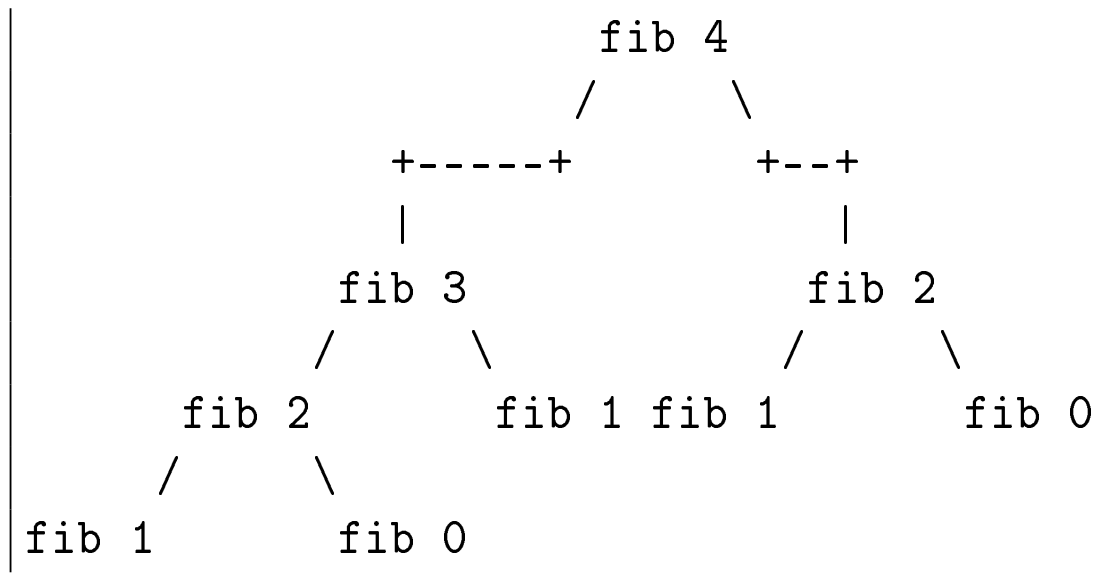
- Inconveniente de la técnica divide y vencerás: la posibilidad de crear idénticos supbroblemas y repetición del trabajo.
- Idea de la programación dinámica: resolver primero los subproblemas menores, guardar los resultados y usar los resultados de los subproblemas intermedios para resolver los mayores.

Cálculo de Fibonacci por divide y vencerás

- Definición de Fibonacci por divide y vencerás.

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

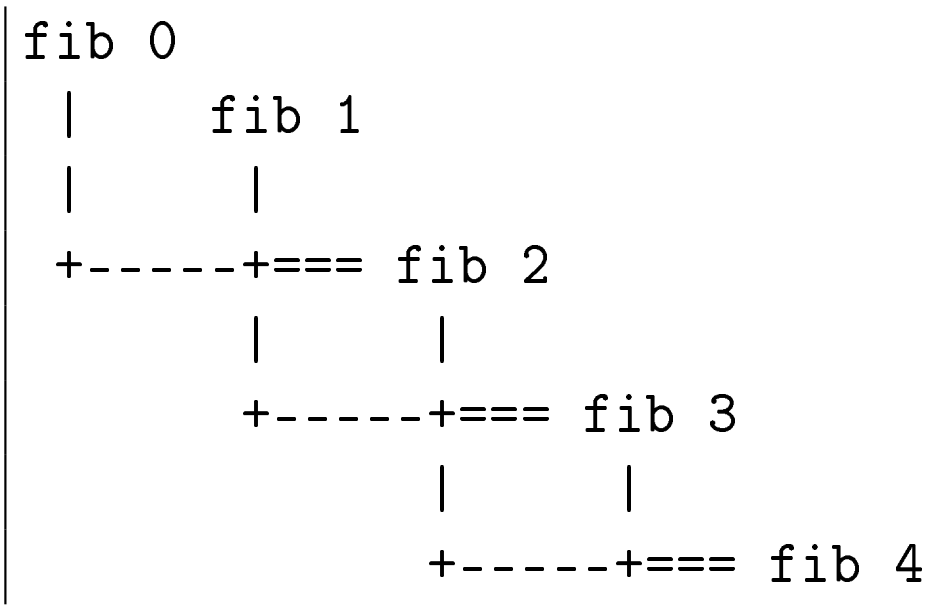
- Cálculo de (fib 4) por divide y vencerás



Calcula 2 veces (fib 2) y 3 veces (fib 1) y (fib 0).

Cálculo de Fibonacci por programación dinámica

- Cálculo de (fib 4) por programación dinámica



Tema 24: Técnicas de diseño ascendente de algoritmos

El patrón de la programación dinámica

- Cabecera del módulo:

```
module Dinamica (module Tabla, dinamica)  where
```

- Librerías auxiliares

```
-- Hay que elegir una implementación de TAD Tabla
-- import TablaConFunciones as Tabla
import TablaConListasDeAsociacion as Tabla
-- import TablaConMatrices as Tabla

import Data.Array
```

El patrón de la programación dinámica

- El patrón de la programación dinámica

```
dinamica :: Ix i => (Tabla i v -> i -> v) -> (i,i)
                    -> Tabla i v
dinamica calcula cotas = t
    where t = tabla [(i,calcula t i) | i <- range cotas]
```

- `(calcula t i)` es el valor del índice `i` calculado a partir de los anteriores que ya se encuentran en la tabla `t`.
- `cotas` son las cotas de la matriz `t` en la que se almacenan los valores calculados.

Tema 24: Técnicas de diseño ascendente de algoritmos

Definición de Fibonacci mediante programación dinámica

- Importación del patrón de programación dinámica

```
import Dinamica
```

- `(fib n)` es el n-ésimo término de la sucesión de Fibonacci, calculado mediante programación dinámica. Por ejemplo,

```
fib 8  ⇝  21
```

```
fib :: Int -> Int
fib n = valor t n
    where t = dinamica calculaFib (cotasFib n)
```

Definición de Fibonacci mediante programación dinámica

- `(calculaFib t i)` es el valor de i-ésimo término de la sucesión de Fibonacci calculado mediante la tabla t que contiene los anteriores. Por ejemplo,

```
calculaFib (tabla []) 0                          ⇝ 0
calculaFib (tabla [(0,0),(1,1),(2,1),(3,2)] 4 ⇝ 3
```

Además,

```
ghci> dinamica calculaFib (0,6)
Tbl [(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)]
```

```
calculaFib :: Tabla Int Int -> Int -> Int
calculaFib t i
  | i <= 1    = i
  | otherwise = valor t (i-1) + valor t (i-2)
```

Definición de Fibonacci mediante programación dinámica

- `(calculaFib t i)` es el valor de i-ésimo término de la sucesión de Fibonacci calculado mediante la tabla t que contiene los anteriores. Por ejemplo,

```
calculaFib (tabla []) 0                          ⇝ 0
calculaFib (tabla [(0,0),(1,1),(2,1),(3,2)] 4 ⇝ 3
```

Además,

```
ghci> dinamica calculaFib (0,6)
Tbl [(0,0),(1,1),(2,1),(3,2),(4,3),(5,5),(6,8)]
```

```
calculaFib :: Tabla Int Int -> Int -> Int
calculaFib t i
   | i <= 1    = i
   | otherwise = valor t (i-1) + valor t (i-2)
```

Definición de Fibonacci mediante programación dinámica

- `(cotasFib n)` son las cotas del vector que se necesita para calcular el n-ésimo término de la sucesión de Fibonacci mediante programación dinámica.

```
cotasFib :: Int -> (Int,Int)
cotasFib n = (0,n)
```

Definición de Fibonacci mediante programación dinámica

- `(cotasFib n)` son las cotas del vector que se necesita para calcular el n-ésimo término de la sucesión de Fibonacci mediante programación dinámica.

```
cotasFib :: Int -> (Int,Int)
cotasFib n = (0,n)
```

Definición de Fibonacci mediante divide y vencerás

- `(fibR n)` es el n–ésimo término de la sucesión de Fibonacci calculado mediante divide y vencerás.

```
fibR :: Int -> Int
fibR 0 = 0
fibR 1 = 1
fibR n = fibR (n-1) + fibR (n-2)
```

- Comparación:

```
ghci> fib 30
832040
(0.01 secs, 0 bytes)
ghci> fibR 30
832040
(6.46 secs, 222602404 bytes)
```

Definición de Fibonacci mediante evaluación perezosa

- `fibs` es la lista de los términos de la sucesión de Fibonacci. Por ejemplo,

```
take 10 fibs  ⇝  [0,1,1,2,3,5,8,13,21,34]
```

```
fibs :: [Int]
fibs = 0:1:[x+y | (x,y) <- zip fibs (tail fibs)]
```

- `(fib' n)` es el n-ésimo término de la sucesión de Fibonacci, calculado a partir de fibs. Por ejemplo,

```
fib' 8  ⇝  21
```

```
fib' :: Int -> Int
fib' n = fibs!!n
```

Definición de Fibonacci mediante evaluación perezosa

- Comparaciones:

```
ghci> fib 30
832040
(0.02 secs, 524808 bytes)
ghci> fib' 30
832040
(0.01 secs, 542384 bytes)
ghci> fibR 30
832040
(6.46 secs, 222602404 bytes)
```

Tema 24: Técnicas de diseño ascendente de algoritmos

Descripción del problema

- Para multiplicar una matriz de orden $m * p$ y otra de orden $p * n$ se necesitan mnp multiplicaciones de elementos.
- El problema del producto de una cadena de matrices (en inglés, "matrix chain multiplication") consiste en dada una sucesión de matrices encontrar la manera de multiplicarlas usando el menor número de productos de elementos.
- Ejemplo: Dada la sucesión de matrices

$$A(30x1), B(1x40), C(40x10), D(10x25)$$

las productos necesarios en las posibles asociaciones son

$((AB)C)D$	$30x1x40$	$+$	$30x40x10$	$+$	$30x10x25$	$=$	20700
$A(B(CD))$	$40x10x25$	$+$	$1x40x25$	$+$	$30x1x25$	$=$	11750
$(AB)(CD)$	$30x1x40$	$+$	$40x10x25$	$+$	$30x40x25$	$=$	41200
$A((BC)D)$	$1x40x10$	$+$	$1x10x25$	$+$	$30x1x25$	$=$	1400
$(A(BC))D$	$1x40x10$	$+$	$30x1x10$	$+$	$30x10x25$	$=$	8200

El algoritmo del PCM

- El PCM correspondiente a la sucesión $d_0, \ldots, d_n$ consiste en encontrar la manera de multiplicar una sucesión de matrices $A_1, \ldots, A_n$ (tal que el orden de A_i es $d_{i-1} \times d_i$) usando el menor número de productos de elementos.
- Sea $c_{i,j}$ el mínimo número de multiplicaciones necesarias para multiplicar la cadena $A_i, \ldots, A_j$ $(1 \leq i \leq j \leq n)$.
- Relación de recurrencia de $c_{i,j}$:

$$\begin{aligned} c_{i,i} &= 0 \\ c_{i,j} &= minimo\{c_{i,k} + c_{k+1,j} + d_{i-1} d_k d_j | i \leq k < j\} \end{aligned}$$

- La solución del problema es $c_{1,n}$.

Tema 24: Técnicas de diseño ascendente de algoritmos

Solución del PCM mediante programación dinámica

- Importación de librerías auxiliares:

```
import Dinamica
```

- `Cadena` representa el producto de una cadena de matrices. Por ejemplo,

```
P (A 1) (P (A 2) (A 3))  ⇝  (A1*(A2*A3))
P (P (A 1) (A 2)) (A 3)  ⇝  ((A1*A2)*A3)
```

```
data Cadena = A Int
            | P Cadena Cadena

instance Show Cadena where
    show (A x)      = "A" ++ show x
    show (P p1 p2) = concat ["(",show p1,"*",show p2,")"]
```

Solución del PCM mediante programación dinámica

- Los índices de la matriz de cálculo son de la forma `(i,j)` y sus valores `(v,k)` donde `v` es el mínimo número de multiplicaciones necesarias para multiplicar la cadena $A_i, \ldots, A_j$ y `k` es la posición donde dividir la cadena de forma óptima.

```
type IndicePCM = (Int,Int)
type ValorPCM  = (Int,Int)
```

Solución del PCM mediante programación dinámica

- `(pcm ds)` es el par formado por el mínimo número de multiplicaciones elementales para multiplicar una sucesión de matrices $A_1, \ldots, A_n$ (tal que el orden de A_i es $d_{i-1} \times d_i$ y $ds = [d_0, \ldots, d_n]$). Por ejemplo,

```
pcm [30,1,40,10,25]  ⇝ (1400,(A1*((A2*A3)*A4)))
```

```
pcm :: [Int] -> (Int, Cadena)
pcm ds = (v, cadena t 1 n)
    where n      = length ds - 1
          t      = dinamica (calculaPCM ds) (cotasPCM n)
          (v,_) = valor t (1,n)
```

Solución del PCM mediante programación dinámica

- `(pcm ds)` es el par formado por el mínimo número de multiplicaciones elementales para multiplicar una sucesión de matrices $A_1, \ldots, A_n$ (tal que el orden de A_i es $d_{i-1} \times d_i$ y $ds = [d_0, \ldots, d_n]$). Por ejemplo,

```
pcm [30,1,40,10,25]  ⇝ (1400,(A1*((A2*A3)*A4)))
```

```
pcm :: [Int] -> (Int, Cadena)
pcm ds = (v, cadena t 1 n)
    where n      = length ds - 1
          t      = dinamica (calculaPCM ds) (cotasPCM n)
          (v,_) = valor t (1,n)
```

Solución del PCM mediante programación dinámica

- `(calculaPCM ds t (i,j))` es el valor del índice `(i,j)` calculado a partir de la lista `ds` de dimensiones de las matrices y la tabla `t` de valores previamente calculados.

```
calculaPCM :: [Int] -> Tabla IndicePCM ValorPCM
              -> IndicePCM -> ValorPCM
calculaPCM ds t (i,j)
    | i == j    = (0,i)
    | otherwise =
        minimum [(fst(valor t (i,k))
                  + fst(valor t (k+1,j))
                  + ds!!(i-1) * ds!!k * ds!!j, k)
                  | k <- [i..j-1]]
```

Solución del PCM mediante programación dinámica

- `(calculaPCM ds t (i,j))` es el valor del índice `(i,j)` calculado a partir de la lista `ds` de dimensiones de las matrices y la tabla `t` de valores previamente calculados.

```
calculaPCM :: [Int] -> Tabla IndicePCM ValorPCM
              -> IndicePCM -> ValorPCM
calculaPCM ds t (i,j)
    | i == j    = (0,i)
    | otherwise =
        minimum [(fst(valor t (i,k))
                  + fst(valor t (k+1,j))
                  + ds!!(i-1) * ds!!k * ds!!j, k)
                  | k <- [i..j-1]]
```

Solución del PCM mediante programación dinámica

- `(cotasPCM n)` son las cotas de los índices para el producto de una cadena de n matrices.

```
cotasPCM :: Int -> (IndicePCM,IndicePCM)
cotasPCM n = ((1,1),(n,n))
```

- `(cadena t i j)` es la cadena que resultar de agrupar las matrices $A_i, \ldots, A_j$ según los valores de la tabla `t`.

```
cadena :: Tabla IndicePCM ValorPCM -> Int -> Int -> Cadena
cadena t i j
    | i == j-1  = P (A i) (A j)
    | k == i    = P (A i) (cadena t (i+1) j)
    | k == j-1  = P (cadena t i (j-1)) (A j)
    | otherwise = P (cadena t i (k-1)) (cadena t k j)
   where (_,k) = valor t (i,j)
```

Solución del PCM mediante programación dinámica

- `(cotasPCM n)` son las cotas de los índices para el producto de una cadena de n matrices.

```
cotasPCM :: Int -> (IndicePCM,IndicePCM)
cotasPCM n = ((1,1),(n,n))
```

- `(cadena t i j)` es la cadena que resultar de agrupar las matrices $A_i, \ldots, A_j$ según los valores de la tabla `t`.

```
cadena :: Tabla IndicePCM ValorPCM -> Int -> Int -> Cadena
cadena t i j
    | i == j-1  = P (A i) (A j)
    | k == i    = P (A i) (cadena t (i+1) j)
    | k == j-1  = P (cadena t i (j-1)) (A j)
    | otherwise = P (cadena t i (k-1)) (cadena t k j)
   where (_,k) = valor t (i,j)
```

Solución del PCM mediante programación dinámica

- `(cotasPCM n)` son las cotas de los índices para el producto de una cadena de n matrices.

```
cotasPCM :: Int -> (IndicePCM,IndicePCM)
cotasPCM n = ((1,1),(n,n))
```

- `(cadena t i j)` es la cadena que resultar de agrupar las matrices $A_i, \ldots, A_j$ según los valores de la tabla `t`.

```
cadena :: Tabla IndicePCM ValorPCM -> Int -> Int -> Cadena
cadena t i j
    | i == j-1  = P (A i) (A j)
    | k == i    = P (A i) (cadena t (i+1) j)
    | k == j-1  = P (cadena t i (j-1)) (A j)
    | otherwise = P (cadena t i (k-1)) (cadena t k j)
    where (_,k) = valor t (i,j)
```

Solución del PCM mediante programación dinámica

- `(pcm' ds)` es la lista de los índices y valores usados en el cálculo del mínimo número de multiplicaciones necesarias para multiplicar una sucesión de matrices $A_1, \ldots, A_n$ (tal que el orden de A_i es $d_{i-1} \times d_i$ y $ds = [d_0, \ldots, d_n]$). Por ejemplo,

```
ghci> pcm' [30,1,40,10,25]
[((1,1),(0,1)),((1,2),(1200,1)),((1,3),(700,1)),((1,4),(1400,1))
 ((2,2),(0,2)),((2,3),(400,2)),((2,4),(650,3)),
 ((3,3),(0,3)),((3,4),(10000,3)),
 ((4,4),(0,4))]
```

```
pcm' :: [Int] -> [((Int, Int), ValorPCM)]
pcm' ds = [((i,j),valor t (i,j)) | i <- [1..n], j <- [i..n]]
    where n = length ds - 1
          t = dinamica (calculaPCM ds) (cotasPCM n)
```

Solución del PCM mediante programación dinámica

- `(pcm' ds)` es la lista de los índices y valores usados en el cálculo del mínimo número de multiplicaciones necesarias para multiplicar una sucesión de matrices $A_1, \ldots, A_n$ (tal que el orden de A_i es $d_{i-1} \times d_i$ y $ds = [d_0, \ldots, d_n]$). Por ejemplo,

```
ghci> pcm' [30,1,40,10,25]
[((1,1),(0,1)),((1,2),(1200,1)),((1,3),(700,1)),((1,4),(1400,1))
 ((2,2),(0,2)),((2,3),(400,2)),((2,4),(650,3)),
 ((3,3),(0,3)),((3,4),(10000,3)),
 ((4,4),(0,4))]
```

```
pcm' :: [Int] -> [((Int, Int), ValorPCM)]
pcm' ds = [((i,j),valor t (i,j)) | i <- [1..n], j <- [i..n]]
    where n = length ds - 1
          t = dinamica (calculaPCM ds) (cotasPCM n)
```

Tema 24: Técnicas de diseño ascendente de algoritmos

Solución del PCM mediante divide y vencerás

- `(pcmDyV ds)` es la solución del PCM correspondiente a `ds` mediante divide y vencerás. Por ejemplo,

  `pcmDyV [30,1,40,10,25]` ⇝ `(1040,(A1*((A2*A3)*A4)))`

```
pcmDyV :: [Int] -> (Int, Cadena)
pcmDyV ds = cadenaDyV ds 1 n
    where n = length ds - 1
```

Solución del PCM mediante divide y vencerás

- `(pcmDyV ds)` es la solución del PCM correspondiente a `ds` mediante divide y vencerás. Por ejemplo,

  `pcmDyV [30,1,40,10,25]` ⇝ `(1040,(A1*((A2*A3)*A4)))`

```
pcmDyV :: [Int] -> (Int, Cadena)
pcmDyV ds = cadenaDyV ds 1 n
    where n = length ds - 1
```

Solución del PCM mediante divide y vencerás

- `cadenaDyV ds i j)` es la solución del PCM correspondiente a $[d_i, \ldots, d_j]$. Por ejemplo,

```
cadenaDyV [30,1,40,10,25] 1 4  ⇝  (1040,(A1*((A2*A3)*A4)))
cadenaDyV [30,1,40,10,25] 2 4  ⇝  (290,((A2*A3)*A4))
```

```
cadenaDyV :: [Int] -> Int -> Int -> (Int, Cadena)
cadenaDyV ds i j
    | i == j     = (0, A i)
    | i == j-1   = (ds!!1*ds!!2, P (A i) (A j))
    | k == i     = (v, P (A i) (subcadena (i+1) j))
    | k == j-1   = (v, P (subcadena i (j-1)) (A j))
    | otherwise = (v, P (subcadena i (k-1)) (subcadena k j))
    where (v,k) = minimum [((valor i k)
                            + (valor (k+1) j)
                            + ds!!(i-1) * ds!!k * ds!!j, k)
                           | k <- [i..j-1]]
          valor p q     = fst (cadenaDyV ds p q)
          subcadena p q = snd (cadenaDyV ds p q)
```

Solución del PCM mediante divide y vencerás

- `cadenaDyV ds i j)` es la solución del PCM correspondiente a $[d_i, \ldots, d_j]$. Por ejemplo,

```
cadenaDyV [30,1,40,10,25] 1 4  ⇝  (1040,(A1*((A2*A3)*A4)))
cadenaDyV [30,1,40,10,25] 2 4  ⇝  (290,((A2*A3)*A4))
```

```
cadenaDyV :: [Int] -> Int -> Int -> (Int, Cadena)
cadenaDyV ds i j
    | i == j     = (0, A i)
    | i == j-1   = (ds!!1*ds!!2, P (A i) (A j))
    | k == i     = (v, P (A i) (subcadena (i+1) j))
    | k == j-1   = (v, P (subcadena i (j-1)) (A j))
    | otherwise = (v, P (subcadena i (k-1)) (subcadena k j))
    where (v,k) = minimum [((valor i k)
                            + (valor (k+1) j)
                            + ds!!(i-1) * ds!!k * ds!!j, k)
                           | k <- [i..j-1]]
          valor p q     = fst (cadenaDyV ds p q)
          subcadena p q = snd (cadenaDyV ds p q)
```

Comparación de las métodos de solucionar el PCM

```
ghci> :set +s

ghci> fst (pcm [1..20])
2658
(0.80 secs, 39158964 bytes)

ghci> fst (pcmDyV [1..20])
1374
(2871.47 secs, 133619742764 bytes)
```

Tema 24: Técnicas de diseño ascendente de algoritmos

Descripción del problema de ABBO

- Para cada clave c_i, sea p_i la probabilidad de acceso a c_i.
- Un árbol binario de búsqueda es optimal (ABBO) si la media del número de comparaciones para todas las claves

$$a(T) = \Sigma d_i p_i$$

donde d_i es la distancia de la clave c_i a la raíz (es decir, el número de comparaciones necesarias para llegar a c_i), es mínima.

El algoritmo del ABBO

- Sea $c_{i,j}$ el mínimo valor $a(T)$ cuando el árbol T contiene las claves $c_i, \ldots, c_j$.
- Relación de recurrencia para calcular $c_{i,j}$:
  - Si $i > j$, $c_{i,j} = 0$.
  - Si $i = j$, $c_{i,j} = p_i$.
  - Si $i < j$,

$$c_{i,j} = min_{i \leq k \leq j}((c_{i,k-1} + \sum_{l=i}^{l=k-1} p_l) + (c_{k+1,j} + \sum_{l=k+1}^{l=j} p_l) + p_k)$$

  - El tercer caso puede simplificarse

$$c_{i,j} = min_{i \leq k \leq j}(c_{i,k-1} + c_{k+1,j}) + \sum_{l=i}^{l=j} p(l)$$

Tema 24: Técnicas de diseño ascendente de algoritmos

Solución del ABBO mediante programación dinámica

- En la matriz de cálculo del ABBO el valor `(v,k)` correspondiente al índice `(i,j)` indica que `v` es el mínimo valor `a(T)` cuando el árbol `T` contiene las claves $c_i, \ldots, c_j$ y que la división óptima se obtiene dividiendo las claves en dos mediante c_k.

```
type Indice = (Int,Int)
type Valor  = (Float,Int)
```

- `(ABB a)` es el tipo de los árboles binarios de búsqueda sobre `a`.

```
data ABB a = Vacio
           | Nodo a (ABB a) (ABB a)
           deriving Show
```

Solución del ABBO mediante programación dinámica

- `(abbo cs ps)` es el par formado por un ABBO correspondiente a la lista de claves `cs` cuyas correspondientes probabilidades de acceso son los elementos de la lista `ps` y por su valor. Por ejemplo,

```
ghci> abbo ejProblema
(Nodo 4 (Nodo 1 Vacio
                (Nodo 3 Vacio Vacio))
        (Nodo 10
              (Nodo 8 Vacio Vacio)
              (Nodo 15
                    (Nodo 11 Vacio Vacio)
                    Vacio)),
2.15)
```

Solución del ABBO mediante programación dinámica

- Definición de `abbo`:

```
abbo :: Problema  -> (ABB Int,Float)
abbo pb = (solucion c t (1,n) , fst (valor t (1,n)))
    where (cs,ps) = pb
          n       = length ps
          c       = listArray (1,n) cs
          p       = listArray (1,n) ps
          t       = dinamica (calcula p) (cotas n)
```

Solución del ABBO mediante programación dinámica

- Definición de abbo:

```
abbo :: Problema  -> (ABB Int,Float)
abbo pb = (solucion c t (1,n) , fst (valor t (1,n)))
    where (cs,ps) = pb
          n       = length ps
          c       = listArray (1,n) cs
          p       = listArray (1,n) ps
          t       = dinamica (calcula p) (cotas n)
```

Solución del ABBO mediante programación dinámica

- `(calcula p t (i,j))` es el valor del índice `(i,j)` donde `p` es el vector de probabilidades y `t` es la tabla calculada hasta el momento.

```
calcula :: Array Int Float -> Tabla Indice Valor
           -> Indice -> Valor
calcula p t (i,j)
    | i > j      = (0.0,0)
    | i == j     = (p!i,i)
    | otherwise = suma1 (minimum [(fst(valor t (i,k-1))
                                   + fst(valor t (k+1,j)),
                                   | k <- [i..j]])
                        (sumaSegmento i j p)
                  where suma1 (x,y) z = (x+z,y)
```

Solución del ABBO mediante programación dinámica

- `(calcula p t (i,j))` es el valor del índice `(i,j)` donde `p` es el vector de probabilidades y `t` es la tabla calculada hasta el momento.

```
calcula :: Array Int Float -> Tabla Indice Valor
           -> Indice -> Valor
calcula p t (i,j)
    | i > j      = (0.0,0)
    | i == j     = (p!i,i)
    | otherwise = suma1 (minimum [(fst(valor t (i,k-1))
                                   + fst(valor t (k+1,j)),
                                   | k <- [i..j]])
                        (sumaSegmento i j p)
                  where suma1 (x,y) z = (x+z,y)
```

Solución del ABBO mediante programación dinámica

- `(sumaSegmento i j p)` es la suma de los valores de los elementos del vector `p` desde la posición `i` a la `j`. Por ejemplo,

```
> sumaSegmento 2 4 (array (1,5)
                          [(i,fromIntegral i/2) | i <- [1..5]])
4.5
```

```
sumaSegmento :: Int -> Int -> Array Int Float -> Float
sumaSegmento i j p = sum [p!l | l <- [i..j]]
```

- `(cotas n)` son las cotas de la matriz revesaria para resolver el problema del árbol de búsqueda minimal óptimo con `n` claves.

```
cotas :: Int -> ((Int,Int),(Int,Int))
cotas n = ((1,0),(n+1,n))
```

Solución del ABBO mediante programación dinámica

- `(sumaSegmento i j p)` es la suma de los valores de los elementos del vector `p` desde la posición `i` a la `j`. Por ejemplo,

```
> sumaSegmento 2 4 (array (1,5)
                          [(i,fromIntegral i/2) | i <- [1..5]])
4.5
```

```
sumaSegmento :: Int -> Int -> Array Int Float -> Float
sumaSegmento i j p = sum [p!l | l <- [i..j]]
```

- `(cotas n)` son las cotas de la matriz revesaria para resolver el problema del árbol de búsqueda minimal óptimo con `n` claves.

```
cotas :: Int -> ((Int,Int),(Int,Int))
cotas n = ((1,0),(n+1,n))
```

Solución del ABBO mediante programación dinámica

- `(sumaSegmento i j p)` es la suma de los valores de los elementos del vector `p` desde la posición `i` a la `j`. Por ejemplo,

```
> sumaSegmento 2 4 (array (1,5)
                          [(i,fromIntegral i/2) | i <- [1..5]])
4.5
```

```
sumaSegmento :: Int -> Int -> Array Int Float -> Float
sumaSegmento i j p = sum [p!l | l <- [i..j]]
```

- `(cotas n)` son las cotas de la matriz revesaria para resolver el problema del árbol de búsqueda minimal óptimo con `n` claves.

```
cotas :: Int -> ((Int,Int),(Int,Int))
cotas n = ((1,0),(n+1,n))
```

Solución del ABBO mediante programación dinámica

- `(solucion cs c (i,j))` es el ABBO correspondiente a las claves `c(i)`,...,`c(j)` a partir de la tabla de cálculo `t`.

```
solucion :: Array Int Int -> Tabla Indice Valor
            -> Indice -> ABB Int
solucion cs t (i,j)
         | i > j     = Vacio
         | i == j    = Nodo c Vacio Vacio
         | otherwise = Nodo c (solucion cs t (i,k-1))
                              (solucion cs t (k+1,j))
         where (_,k) = valor t (i,j)
               c     = cs ! k
```

Solución del ABBO mediante programación dinámica

- `(solucion cs c (i,j))` es el ABBO correspondiente a las claves `c(i)`,...,`c(j)` a partir de la tabla de cálculo `t`.

```
solucion :: Array Int Int -> Tabla Indice Valor
            -> Indice -> ABB Int
solucion cs t (i,j)
        | i > j      = Vacio
        | i == j     = Nodo c Vacio Vacio
        | otherwise = Nodo c (solucion cs t (i,k-1))
                             (solucion cs t (k+1,j))
        where (_,k) = valor t (i,j)
              c     = cs ! k
```

Tema 24: Técnicas de diseño ascendente de algoritmos

Descripción del problema

- Cálculo de los caminos de coste mínimo entre todos los pares de nodos de un grafo no dirigido.
- Notación:
  - $c_{i,j}$ es el mínimo coste del camino del vértice i al j.
  - $p_{i,j} = \begin{cases} 0, & \text{si } i = j \\ \text{peso del arco entre } i \text{ y } j, & \text{si } i \neq j \text{ y hay arco de } i \text{ a } j \\ \infty, & \text{en otro caso} \end{cases}$
  - $c_{i,j,k}$ es el mínimo coste del camino del vértice i al j, usando los vértices $1, \ldots, k$.
- Relación de recurrencia para calcular $c_{i,j}$:
  - $c_{i,j,0} = p_{i,j}$
  - $c_{i,j,k} = min\{c_{i,j,k-1)}, c_{i,k,k-1} + c_{k,j,k-1}\}$
- El algoritmo se conoce como el algoritmo de Floyd.

Tema 24: Técnicas de diseño ascendente de algoritmos

Solución del CM mediante programación dinámica

- Importación de librerías auxiliares:

```
import Dinamica

-- Nota: Elegir una implementación de los grafos.
import GrafoConVectorDeAdyacencia
-- import GrafoConMatrizDeAdyacencia
```

Solución del CM mediante programación dinámica

- Ejemplos de grafos para el problema:

```
ej1Grafo :: Grafo Int Int
ej1Grafo = creaGrafo True (1,6)
                          [(i,j,(v!!(i-1))!!(j-1))
                           | i <- [1..6], j <- [1..6]]


v::[[Int]]
v = [[  0,  4,  1,  6,100,100],
     [  4,  0,  1,100,  5,100],
     [  1,  1,  0,100,  8,  2],
     [  6,100,100,  0,100,  2],
     [100,  5,  8,100,  0,  5],
     [100,100,  2,  2,  5,  0]]
```

Solución del CM mediante programación dinámica

- Ejemplos de grafos para el problema:

```
ej2Grafo :: Grafo Int Int
ej2Grafo = creaGrafo True (1,6)
                          [(i,j,(v'!!(i-1))!!(j-1))
                          | i <- [1..6], j <- [1..6]]


v'::[[Int]]
v' =[[  0,  4,100,100,100,  2],
     [  1,  0,  3,  4,100,100],
     [  6,  3,  0,  7,100,100],
     [  6,100,100,  0,  2,100],
     [100,100,100,  5,  0,100],
     [100,100,100,  2,  3,  0]]
```

Solución del CM mediante programación dinámica

- En la matriz del cálculo del camino mínimo, los índices son de la forma `(i,j,k)` y los valores de la forma `(v,xs)` representando que el camino mínimo desde el vértice `i` al `j` usando los vértices `1`, ..., `k` tiene un coste `v` y está fomado por los vértices `xs`.

```
type IndiceCM = (Int,Int,Int)
type ValorCM  = (Int,[Int])
```

Solución del CM mediante programación dinámica

- `(caminosMinimos g)` es la lista de los caminos mínimos entre todos los nodos del grafo g junto con sus costes. Por ejemplo,

```
ghci> caminosMinimos ej1Grafo
[((1,2),(2,[1,3,2])),  ((1,3),(1,[1,3])),  ((1,4),(5,[1,3,6,4]))
 ((1,5),(7,[1,3,2,5])),((1,6),(3,[1,3,6])),((2,3),(1,[2,3])),
 ((2,4),(5,[2,3,6,4])),((2,5),(5,[2,5])),  ((2,6),(3,[2,3,6])),
 ((3,4),(4,[3,6,4])),  ((3,5),(6,[3,2,5])),((3,6),(2,[3,6])),
 ((4,5),(7,[4,6,5])),  ((4,6),(2,[4,6])),  ((5,6),(5,[5,6]))]
```

```
caminosMinimos :: (Grafo Int Int) -> [((Int,Int), ValorCM)]
caminosMinimos g =
    [((i,j), valor t (i,j,n)) | i <- [1..n], j <- [i+1..n]]
    where n = length (nodos g)
          t = dinamica (calculaCM g) (cotasCM n)
```

Solución del CM mediante programación dinámica

- `(caminosMinimos g)` es la lista de los caminos mínimos entre todos los nodos del grafo g junto con sus costes. Por ejemplo,

```
ghci> caminosMinimos ej1Grafo
[((1,2),(2,[1,3,2])),  ((1,3),(1,[1,3])),  ((1,4),(5,[1,3,6,4]))
 ((1,5),(7,[1,3,2,5])),((1,6),(3,[1,3,6])),((2,3),(1,[2,3])),
 ((2,4),(5,[2,3,6,4])),((2,5),(5,[2,5])),  ((2,6),(3,[2,3,6])),
 ((3,4),(4,[3,6,4])),  ((3,5),(6,[3,2,5])),((3,6),(2,[3,6])),
 ((4,5),(7,[4,6,5])),  ((4,6),(2,[4,6])),  ((5,6),(5,[5,6]))]
```

```
caminosMinimos :: (Grafo Int Int) -> [((Int,Int), ValorCM)]
caminosMinimos g =
    [((i,j), valor t (i,j,n)) | i <- [1..n], j <- [i+1..n]]
    where n = length (nodos g)
          t = dinamica (calculaCM g) (cotasCM n)
```

Solución del CM mediante programación dinámica

- `(calculaCM g t (i,j,k))` es el valor del camino mínimo desde el vértice `i` al `j` usando los vértices `1`, ..., `k` del grafo `g` y la tabla `t` de los valores anteriores al índice `(i,j,k)`.

```
calculaCM :: (Grafo Int Int) -> Tabla IndiceCM ValorCM
             -> IndiceCM -> ValorCM
calculaCM g t (i,j,k)
  | k==0      = (peso i j g, if i==j then [i] else [i,j])
  | v1<=v2    = (v1,p)
  | otherwise = (v2,p1++p2)
  where (v1,p)   = valor t (i,j,k-1)
        (a,p1)   = valor t (i,k,k-1)
        (b,_:p2) = valor t (k,j,k-1)
        v2 = a+b
```

Solución del CM mediante programación dinámica

- `(calculaCM g t (i,j,k))` es el valor del camino mínimo desde el vértice `i` al `j` usando los vértices `1`, ..., `k` del grafo `g` y la tabla `t` de los valores anteriores al índice `(i,j,k)`.

```
calculaCM :: (Grafo Int Int) -> Tabla IndiceCM ValorCM
             -> IndiceCM -> ValorCM
calculaCM g t (i,j,k)
  | k==0      = (peso i j g, if i==j then [i] else [i,j])
  | v1<=v2    = (v1,p)
  | otherwise = (v2,p1++p2)
  where (v1,p)   = valor t (i,j,k-1)
        (a,p1)   = valor t (i,k,k-1)
        (b,_:p2) = valor t (k,j,k-1)
        v2 = a+b
```

Solución del CM mediante programación dinámica

- `(cotasCM n)` son las cotas de la matriz para resolver el problema de los caminos mínimos en un grafo con `n` nodos.

```
cotasCM :: Int -> ((Int,Int,Int),(Int,Int,Int))
cotasCM n = ((1,1,0),(n,n,n))
```

Solución del CM mediante programación dinámica

- `(cotasCM n)` son las cotas de la matriz para resolver el problema de los caminos mínimos en un grafo con `n` nodos.

```
cotasCM :: Int -> ((Int,Int,Int),(Int,Int,Int))
cotasCM n = ((1,1,0),(n,n,n))
```

Tema 24: Técnicas de diseño ascendente de algoritmos

Descripción del problema

- Dado un grafo no dirigido con pesos encontrar una camino en el grafo que visite todos los nodos exactamente una vez y cuyo coste sea mínimo.
- Notación:
  - Los vértices del grafo son $1, 2, \ldots, n$.
  - $p_{i,j} = \begin{cases} 0, & \text{si } i = j \\ \text{peso del arco entre } i \text{ y } j, & \text{si } i \neq j \text{ y hay arco de } i \text{ a } j \\ \infty, & \text{en otro caso} \end{cases}$
  - El vértice inicial y final es el n.
  - $c_{i,S}$ es el camino más corto que comienza en i, termina en n y pasa exactamente una vez por cada uno de los vértices del conjunto S.
- Relación de recurrencia de $c_{i,S}$:
  - $c_{i,\emptyset} = p_{i,n}$, si $i \neq n$.
  - $c_{i,S} = min\{p_{i,j} + c_{j,S-\{j\}} : j \in S\}$, si $i \neq n$, $i \notin S$.
- La solución es $c_{n,\{1,\ldots,n-1\}}$.

Tema 24: Técnicas de diseño ascendente de algoritmos

Solución del PV mediante programación dinámica

- Importación de librerías auxiliares

```
import Dinamica

-- Nota: Elegir una implementación de los grafos.
import GrafoConVectorDeAdyacencia
-- import GrafoConMatrizDeAdyacencia
```

- Nota: Para el PV se usará la representación de los de conjuntos de enteros como números enteros que se describe a continuación.
- Los conjuntos se representan por números enteros.

```
type Conj = Int
```

Solución del PV mediante programación dinámica

- `(conj2Lista c)` es la lista de los elementos del conjunto c. Por ejemplo,

```
conj2Lista 24  ⇝  [3,4]
conj2Lista 30  ⇝  [1,2,3,4]
conj2Lista 22  ⇝  [1,2,4]
```

```
conj2Lista :: Conj -> [Int]
conj2Lista s = c2l s 0
    where
      c2l 0 _                = []
      c2l n i | odd n        = i : c2l (n `div` 2) (i+1)
              | otherwise = c2l (n `div` 2) (i+1)
```

Solución del PV mediante programación dinámica

- `(conj2Lista c)` es la lista de los elementos del conjunto c. Por ejemplo,

```
conj2Lista 24  ⇝  [3,4]
conj2Lista 30  ⇝  [1,2,3,4]
conj2Lista 22  ⇝  [1,2,4]
```

```
conj2Lista :: Conj -> [Int]
conj2Lista s = c2l s 0
    where
      c2l 0 _                = []
      c2l n i | odd n        = i : c2l (n `div` 2) (i+1)
              | otherwise    = c2l (n `div` 2) (i+1)
```

Solución del PV mediante programación dinámica

- `maxConj` es el máximo número que puede pertenecer al conjunto. Depende de la implementación de Haskell.

```
maxConj :: Int
maxConj =
   truncate (logBase 2 (fromIntegral maxInt)) - 1
   where maxInt = maxBound::Int
```

- `vacio` es el conjunto vacío.

```
vacio :: Conj
vacio = 0
```

Solución del PV mediante programación dinámica

- `maxConj` es el máximo número que puede pertenecer al conjunto. Depende de la implementación de Haskell.

```
maxConj :: Int
maxConj =
   truncate (logBase 2 (fromIntegral maxInt)) - 1
   where maxInt = maxBound::Int
```

- `vacio` es el conjunto vacío.

```
vacio :: Conj
vacio = 0
```

Solución del PV mediante programación dinámica

- `maxConj` es el máximo número que puede pertenecer al conjunto. Depende de la implementación de Haskell.

```
maxConj :: Int
maxConj =
   truncate (logBase 2 (fromIntegral maxInt)) - 1
   where maxInt = maxBound::Int
```

- `vacio` es el conjunto vacío.

```
vacio :: Conj
vacio = 0
```

Solución del PV mediante programación dinámica

- `(esVacio c)` se verifica si c es el conjunto vacío.

```
esVacio :: Conj -> Bool
esVacio n = n==0
```

- `(conjCompleto n)` es el conjunto de los números desde 1 hasta n.

```
conjCompleto :: Int -> Conj
conjCompleto n
   | (n>=0) && (n<=maxConj) = 2^(n+1)-2
   | otherwise = error ("conjCompleto:" ++ show n)
```

Solución del PV mediante programación dinámica

- `(esVacio c)` se verifica si c es el conjunto vacío.

```
esVacio :: Conj -> Bool
esVacio n = n==0
```

- `(conjCompleto n)` es el conjunto de los números desde 1 hasta n.

```
conjCompleto :: Int -> Conj
conjCompleto n
   | (n>=0) && (n<=maxConj) = 2^(n+1)-2
   | otherwise = error ("conjCompleto:" ++ show n)
```

Solución del PV mediante programación dinámica

- `(esVacio c)` se verifica si c es el conjunto vacío.

```
esVacio :: Conj -> Bool
esVacio n = n==0
```

- `(conjCompleto n)` es el conjunto de los números desde 1 hasta n.

```
conjCompleto :: Int -> Conj
conjCompleto n
   | (n>=0) && (n<=maxConj) = 2^(n+1)-2
   | otherwise = error ("conjCompleto:" ++ show n)
```

Solución del PV mediante programación dinámica

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto `c`.

```
inserta :: Int -> Conj -> Conj
inserta i s
    | i>=0 && i<=maxConj = d'*e+m
    | otherwise         = error ("inserta:" ++ show i)
   where (d,m) = divMod s e
         e     = 2^i
         d'    = if odd d then d else d+1
```

Solución del PV mediante programación dinámica

- `(inserta x c)` es el conjunto obtenido añadiendo el elemento `x` al conjunto c.

```
inserta :: Int -> Conj -> Conj
inserta i s
    | i>=0 && i<=maxConj = d'*e+m
    | otherwise          = error ("inserta:" ++ show i)
    where (d,m) = divMod s e
          e     = 2^i
          d'    = if odd d then d else d+1
```

Solución del PV mediante programación dinámica

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`.

```
elimina :: Int -> Conj -> Conj
elimina i s = d'*e+m
    where (d,m) = divMod s e
          e = 2^i
          d' = if odd d then d-1 else d
```

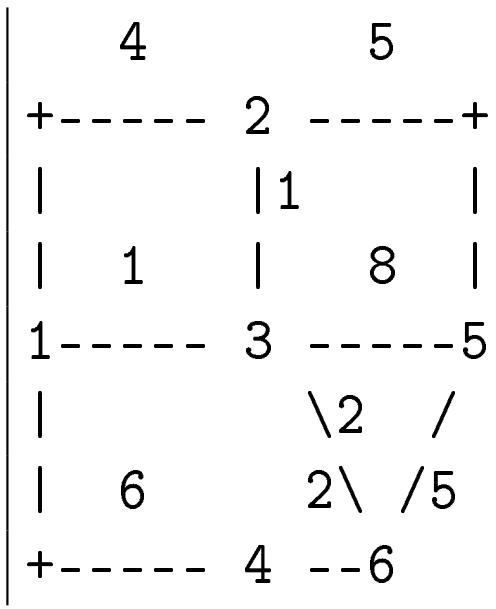
Solución del PV mediante programación dinámica

- `(elimina x c)` es el conjunto obtenido eliminando el elemento `x` del conjunto `c`.

```
elimina :: Int -> Conj -> Conj
elimina i s = d'*e+m
    where (d,m) = divMod s e
          e = 2^i
          d' = if odd d then d-1 else d
```

Solución del PV mediante programación dinámica

- Ejemplo de grafo para el problema:



Solución del PV mediante programación dinámica

- La definición del grafo anterior es

```
ej1 :: Grafo Int Int
ej1 = creaGrafo True (1,6)
                     [(i,j,(v1!!(i-1))!!(j-1))
                      | i <- [1..6], j <- [1..6]]
v1::[[Int]]
v1 =[[  0,  4,  1,  6,100,100],
     [  4,  0,  1,100,  5,100],
     [  1,  1,  0,100,  8,  2],
     [  6,100,100,  0,100,  2],
     [100,  5,  8,100,  0,  5],
     [100,100,  2,  2,  5,  0]]
```

Solución del PV mediante programación dinámica

- Los índices de la matriz de cálculo son de la forma `(i,S)` y sus valores `(v,xs)` donde `xs` es el camino mínimo desde `i` hasta `n` visitando cada vértice de `S` exactamente una vez y `v` es el coste de `xs`.

```
type IndicePV = (Int,Conj)
type ValorPV  = (Int,[Int])
```

Solución del PV mediante programación dinámica

- `(viajante g)` es el par `(v,xs)` donde `xs` es el camino de menor coste que pasa exactamente una vez por todos los nodos del grafo `g` empezando en su último nodo y `v` es su coste. Por ejemplo,

```
ghci> viajante ej1
(20,[6,4,1,3,2,5,6])
```

```
viajante :: Grafo Int Int -> (Int,[Int])
viajante g = valor t (n,conjCompleto (n-1))
    where n = length (nodos g)
          t = dinamica (calculaPV g n) (cotasPV n)
```

Solución del PV mediante programación dinámica

- `(viajante g)` es el par `(v,xs)` donde `xs` es el camino de menor coste que pasa exactamente una vez por todos los nodos del grafo `g` empezando en su último nodo y `v` es su coste. Por ejemplo,

```
ghci> viajante ej1
(20,[6,4,1,3,2,5,6])
```

```
viajante :: Grafo Int Int -> (Int,[Int])
viajante g = valor t (n,conjCompleto (n-1))
    where n = length (nodos g)
          t = dinamica (calculaPV g n) (cotasPV n)
```

Solución del PV mediante programación dinámica

- `(calculaPV g n t (i,k))` es el valor del camino mínimo en el grafo `g` desde `i` hasta `n`, calculado usando la tabla `t`, visitando cada nodo del conjunto `k` exactamente una vez.

```
calculaPV :: Grafo Int Int -> Int -> Tabla IndicePV ValorPV
          -> IndicePV -> ValorPV
calculaPV g n t (i,k)
    | esVacio k = (peso i n g,[i,n])
    | otherwise = minimum [sumaPrim (valor t (j, elimina j k))
                                    (peso i j g)
                           | j <- conj2Lista k]
    where sumaPrim (v,xs) v' = (v+v',i:xs)
```

Solución del PV mediante programación dinámica

- `(calculaPV g n t (i,k))` es el valor del camino mínimo en el grafo `g` desde `i` hasta `n`, calculado usando la tabla `t`, visitando cada nodo del conjunto `k` exactamente una vez.

```
calculaPV :: Grafo Int Int -> Int -> Tabla IndicePV ValorPV
          -> IndicePV -> ValorPV
calculaPV g n t (i,k)
    | esVacio k = (peso i n g,[i,n])
    | otherwise = minimum [sumaPrim (valor t (j, elimina j k))
                                    (peso i j g)
                           | j <- conj2Lista k]
    where sumaPrim (v,xs) v' = (v+v',i:xs)
```

Solución del PV mediante programación dinámica

- `(cotasPV n)` son las cotas de la matriz de cálculo del problema del viajante en un grafo con `n` nodos.

```
cotasPV :: Int -> ((Int,Conj),(Int,Conj))
cotasPV n = ((1,vacio),(n,conjCompleto n))
```

Solución del PV mediante programación dinámica

- `(cotasPV n)` son las cotas de la matriz de cálculo del problema del viajante en un grafo con `n` nodos.

```
cotasPV :: Int -> ((Int,Conj),(Int,Conj))
cotasPV n = ((1,vacio),(n,conjCompleto n))
```