Práctico 11 - AYED2

Lautaro Bachmann

Contents

Notas

Notacion para cuantificar

Escribir:

$$min_{q \in \{0,1,\ldots,j/d_i\}}(q + cambio(i-1, j - q * d_i)$$

Es equivalente a:

```
mimin := infinito
for q := 0 to (j / d[i]) do
  mimin :=   mimin ‘min‘ (q + cambio(i-1, j-q*$d_i$)
od
```

1)

Como resolver

1. Determinar argumentos de la funcion
2. Declarar variables.
3. Cargar tabla con casos base
4. Implementar *caso* recursivo

Resolucion

```
fun cambio(denom: array[1..n] of nat, monto: nat) ret r : nat
    {- Declaracion de variables -}
    var tabla: array[0..n,0..monto] of nat
    var minimo: nat
    {- Casos base -}
    for i := 0 to n do
        tabla[i,0] := 0
    od
    for j := 1 to monto do
        tabla[0,j] := $\infty$
    od
    {- Caso recursivo -}
    for i := 1 to n do
        for j := 1 to monto do
            minimo := $\infty$
            for q := 0 to (j/denom[i]) do
                minimo:= min(minimo, q + tabla[i-1, j- q*denom[i]])
            od
            tabla[i,j] := minimo
        od
    od
    r:= tabla[n, monto]
end fun
```

2)

Como resolver

Para determinar la “direccion” en la que debe ser cargada la tabla hay que observar a que valores accede el caso recursivo.

Si el caso recursivo accede a la fila anterior, la tabla debe de ser cargada de arriba hacia abajo.

Algo similar pasa con el caso de determinar si la tabla se debe de cargar de derecha a izquierda y viceversa.

Resolucion

La tabla puede ser escrita de derecha a izquiera, sin embargo no puede ser escrita de abajo hacia arriba, ya que en el caso recursivo de la funcion siempre se utilizan elementos de la fila anterior

Programa modificado

```
fun cambio(denom: array[1..n] of nat, monto: nat) ret r : nat
   {- Declaracion de variables -}
   var tabla: array[0..n,0..monto] of nat
   var minimo: nat
   {- Casos base -}
   for i := 0 to n do
      tabla[i,0] := 0
   od
   for j := monto downto 1 do
      tabla[0,j] := $\infty$
   od
   {- Caso recursivo -}
   for i := 1 to n do
      for j := monto downto 1 do
         minimo := $\infty$
         for q := 0 to (j/denom[i]) do
            minimo:= min(minimo, q + tabla[i-1, j- q*denom[i]])
         od
         tabla[i,j] := minimo
      od
   od
   r:= tabla[n, monto]
end fun
```

3)

a)

```
fun cambio(denom: array[1..n] of nat, monto: nat) ret r : nat
    {- Declaracion de variables -}
    var tabla: array[0..n,0..monto] of nat
    var minimo: nat
    var i’: nat
    {- Casos base -}
    for i := 0 to n do
        tabla[i,0] = 0
    od
    {- Caso recursivo -}
    for i := 1 to n do
        for j := 1 to monto do
            minimo:= $\infty$
            i’:= 1
            while d[i’] $\leq$ monto do
                minimo:= min(minimo, tabla[i’,j-d[i’]])
                i’:= i’+1
            od
            tabla[i,j]:= 1 + minimo
        od
    od
    r:= tabla[n, monto]
end fun
```

b)

```
fun cambio(denom: array[1..n] of nat, monto: nat) ret r : nat
  {- Declaracion de variables -}
  var tabla: array[0..n,0..monto] of nat
  var minimo: nat
  {- Casos base -}
  for i := 0 to n do
    tabla[i,0] = 0
  od
  for j := 1 to monto do
    tabla[n,j] := $\infty$
  od
  {- Caso recursivo -}
  for i := n-1 downto 1 do
    for j := 1 to monto do
      if d[i] > j then
        tabla[i,j] := tabla[i+1,j]
      else
tabla[i,j]:= min(tabla[i+1,j],1+tabla[i,j-denom[i]])
      fi
    od
  od
  r:= tabla[n, monto]
end fun
```

4)

3)

Version Normal

```
fun harina(m: array[1..n] of nat, h: array[1..n] of nat, H: nat) ret r : nat
  {- Declaracion de variables -}
  var tabla: array[1..n,0..H] of nat
  var maximo: nat
  {- Casos base -}
  for i := 1 to n do
    tabla[i,0] := 0
  od
  for j := 1 to H do
    if $j \geq h[n]$then
      tabla[n,j]:= m[n]
    else
      tabla[n,j]:= 0
    fi
  od
  {- Caso recursivo -}
  for i := n-1 downto 1 do
    for j := 1 to H do
      maximo:= tabla[i+1, j]
      if $j \geq h[i]$ then
        maximo:= max(maximo, m[i] + tabla[i+1, j-h[i]])
      fi
      tabla[i,j] := maximo
    od
  od
  r:= tabla[1, H]
end fun
```

Tabla Casos base

0	?	?	?
0	?	?	?
0	Inicio	?	?
0	m[n,1]	0	m[n,3]

Version con solucion (CORREGIR)

```
fun harina(m: array[1..n] of nat, h: array[1..n] of nat, H: nat) ret r : List of nat
    {- Declaracion de variables -}
    var tabla: array[1..n,0..H] of nat
    var solucion: array[1..n,0..H] of (List of nat)
    var maximo: nat
    var ind_j: nat
    {- Casos base -}
    for i := 1 to n do
        tabla[i,0] := 0
        solucion[i,0]:= empty_list()
    od
    for j := 1 to H do
        if j $\geq$ h[n]then
            tabla[n,j]:= m[n]
            solucion[n,j]:= empty_list()
            list_addl(solucion[n,j], n)
        else
            tabla[n,j]:= 0
            solucion[n,j]:= empty_list()
        fi
    od
    {- Caso recursivo -}
    for i := n-1 downto 1 do
        for j := 1 to H do
            maximo:= tabla[i+1, j]
            ind_j:= j
            if j $\geq$ h[i] $\wedge$ m[i] + tabla[i+1, j-h[i]] > maximo then
                maximo:= m[i] + tabla[i+1, j-h[i]]
                ind_j:= j- h[i]
            fi
            tabla[i,j] := maximo
            solucion[i,j]:= armar_solucion(solucion[i+1, ind_j], i)
        od
    od
    r:= solucion[1, H]
end fun
```

Ejecucion manual (TERMINAR)

m:= [1,5,3]

h:= [2,3,6]

H:= 6

n:= 3

Casos base:

tabla:=

0					
0					
0					

tabla:=

0						
0						
0	1<6 => 0	2<6 => 0	3<6 => 0	4<6 => 0	5<6 => 0	6=6 => 3

tabla:=

0						
0						
0	0	0	0	0	0	3

solucion:=

[]						
[]						
[]	[]	[]	[]	[]	[]	[3]

Caso recursivo:

4)

```
fun globo(v: array[1..n] of nat, p: array[1..n] of nat, P: nat) ret r : nat
  {- Declaracion de variables -}
  var tabla: array[0..n,0..P] of nat
  {- Casos base -}
  for i := 1 to n do
    tabla[i,0] := 0
  od
  for j := 1 to P do
    tabla[0,j]:= $\infty$
  od
  {- Caso recursivo -}
  for i := 1 to n do
    for j := 1 to P do
      tabla[i,j] := min(v[i] + tabla[i-1,j-p[i]], tabla[i-1,j])
    od
  od
  r:= tabla[n, P]
end fun
```

5)

Version normal

```
fun telefono(m: array[1..n] of nat,
  r: array[1..n] of nat,
  p: array[1..n] of nat,
  ultimo_d: nat) ret r : nat
  {- Declaracion de variables -}
  var tabla: array[0..ultimo_d] of nat
  var ultima_p: nat
  for i := 1 to n do
    ultima_p:= max(ultima_p, p[i])
  od
  {- Casos base -}
  for i := ultima_p+1 to ultimo_d do
    tabla[i]:= 0
  od
  {- Caso recursivo -}
  for d := ultima_p downto 1 do
    maximo:= 0
    for i := 1 to n do
      if p[i] = d then
        maximo:= max(maximo, m[i] * (r[i] - p[i] + 1) + tabla[r[i]+1])
      fi
    od
    tabla[d] := max(tabla[d+1], maximo)
  od
  r:= tabla[0]
end fun
```

Version con solucion (COMPLETAR)

```
fun telefono(m: array[1..n] of nat,
   r: array[1..n] of nat,
   p: array[1..n] of nat,
   ultimo_d: nat) ret r : nat
   {- Declaracion de variables -}
   var tabla: array[0..ultimo_d] of nat
   var ultima_p: nat
   for i := 1 to n do
      ultima_p:= max(ultima_p, p[i])
   od
   {- Casos base -}
   for i := ultima_p+1 to ultimo_d do
      tabla[i]:= 0
   od
   {- Caso recursivo -}
   for d := ultima_p downto 1 do
      maximo:= 0
      for i := 1 to n do
         if p[i] = d then
            maximo:= max(maximo, m[i] * (r[i] - p[i] + 1) + tabla[r[i]+1])
         fi
      od
      tabla[d] := max(tabla[d+1], maximo)
   od
   r:= tabla[0]
end fun
```

6)

```
fun prima(vs: array[1..n] of nat,
          as: array[1..n] of nat,
          bs: array[1..n] of nat,
          A: nat,
          B: nat
  ) ret r : type
  {- Declaracion de variables -}
  tabla: array[0..n,0..A,0..B] of nat
  {- Casos base -}
  for a := 0 to A do
    for b := 0 to B do
      tabla[0, a, b] := 0
    od
  od
  for i := 0 to n do
    tabla[i, 0, 0] := 0
  od
  {- Caso recursivo -}
  for i := 1 to n do
    for a := 1 to A do
      for b := 1 to B do
          if a < as[i] ∨ b < bs[i]then
            tabla[i, a, b] := tabla[i-1, a, b]
          else
            tabla[i, a, b] := max(tabla[i-1, a, b],
                                  vs[i] + tabla[i-1,a-as[i],b-bs[i]]
                              )
          fi
      od
    od
  od
  r:= tabla[n,A,B]
end fun
```

7)

Version normal

```
fun mochilas(v: array[1..n] of nat,
             w: array[1..n] of nat,
             W1: nat,
             W2: nat
             ) ret r : type
  {- Declaracion de variables -}
  var tabla: array[0..n,0..W1,0..W2] of int
  {- Casos base -}
  for w1 := 0 to A do
    for w2 := 0 to B do
      tabla[0, w1, w2] := 0
    od
  od
  for i := 0 to n do
    tabla[i, 0, 0] := 0
  od
  {- Caso recursivo -}
  for i := 1 to n do
    for w1 := 1 to W1 do
      for w2 := 1 to W2 do
        maximo := tabla[i - 1, w1, w2]
        if w[i] $\leq$ w1then
          maximo:= max(v[i] + tabla[i - 1, w1 - w[i], w2], maximo)
        else if w[i] $\leq$ w2 then
          maximo:= max(v[i] + tabla[i - 1, w1, w2 - w[i]], maximo)
        fi
        tabla[i, w1, w2]:= maximo
      od
    od
  od
  r:= tabla[n, W1, W2]
end fun
```

Version con solucion (COMPLETAR)

```
fun mochilas(v: array[1..n] of nat,
             w: array[1..n] of nat,
             W1: nat,
             W2: nat
             ) ret r : type
  {- Declaracion de variables -}
  var tabla: array[0..n,0..W1,0..W2] of int
  {- Casos base -}
  for w1 := 0 to A do
    for w2 := 0 to B do
      tabla[0, w1, w2] := 0
    od
  od
  for i := 0 to n do
    tabla[i, 0, 0] := 0
  od
  {- Caso recursivo -}
  for i := 1 to n do
    for w1 := 1 to W1 do
      for w2 := 1 to W2 do
        maximo := tabla[i - 1, w1, w2]
        if w[i] $\leq$ w1then
          maximo:= max(v[i] + tabla[i - 1, w1 - w[i], w2], maximo)
        else if w[i] $\leq$ w2 then
          maximo:= max(v[i] + tabla[i - 1, w1, w2 - w[i]], maximo)
        fi
        tabla[i, w1, w2]:= maximo
      od
    od
  od
  r:= tabla[n, W1, W2]
end fun
```

8)

```
fun automoviles(a: array[1..2,1..n] of nat,
                t: array[1..2,1..n] of nat
) ret r : type
  {- Declaracion de variables -}
  tabla: array[1..2,1..n] of nat
  var minimo: nat
  {- Casos base -}
  for i := 1 to 2 do
    tabla[i, n]:= a[i,n]
  od
  {- Caso recursivo -}
  for i := 1 to 2 do
    for j := n-1 to 1 do
      minimo:= a[i,j] + tabla[i,j+1]
      if i = 1 then
        minimo:= min(minimo,a[i,j] + t[i,j] + tabla[i+1, j+1])
      else
        minimo:= min(minimo,a[i,j] + t[i,j] + tabla[i-1, j+1])
      fi
      tabla[i,j] := minimo
    od
  od
  r:= min(tabla[1,1], tabla[2,1])
end fun
```

9)

```
fun maxUp(c: array[1..n,1..n] of nat) ret r : nat
    {- Declaracion de variables -}
    tabla: array[1..2,1..n] of nat
    var maximo: nat
    {- Casos base -}
    for j := 1 to n do
        tabla[n, j]:= c[n, j]
    od
    {- Caso recursivo -}
    for i := n-1 to 1 do
        for j := n to 1 do
            tabla[i,j]:= c[i,j] + tabla[i+1,j]
                max c[i,j] + tabla[i+1, max(j-1, 0)]
                max c[i,j] + tabla[i+1, min(n, j+1)]
        od
    od
    r:= 0
    for j := 1 to n do
        r:= max(r, maxUp(1,j))
    od
end fun
```