ISIS 1105 Diseño de Algoritmos
Tarea No. 2
Desarrollo de algoritmos correctos
Métodos Generales de solución de problemas
Programación dinámica
Trabajo por parejas
Entrega: Miércoles, Octubre 19, 12 m. (mediodía)

1 *Máximo común divisor y Mínimo común múltiplo*

Desarrolle un algoritmo que satisfaga la siguiente especificación:

```
[Ctx C: x=A ∧ y=B ∧ A>0 ∧ B>0
 {Pre Q: true}
 INIC;
 {Inv P: x>0 ∧ y>0 ∧ x*u+y*v = 2*A*B ∧ x mcd y = A mcd B}
 {cota t: x+y}
 do … od;
 S1;
 {Pos R: x = A mcd B  ∧  y = A mcm B}
]
```

```
[Ctx C: x=A ∧ y=B ∧ A>0 ∧ B>0
 {Pre Q: true}

  u,v:= B,A;

 {Inv P: x>0 ∧ y>0 ∧ x*u+y*v = 2*A*B ∧ x mcd y = A mcd B}
 {cota t: x+y}

 do x≠y →  if x>y      →    x,v:= x-y,u+v
           [] x<y      →    y,u:= y-x,u+v
           fi
 od;

 {R1: x>0 ∧ y>0 ∧ x*(u+v) = 2*A*B  ∧  x = A mcd B}
 {R2: x>0 ∧ y>0 ∧ (A mcd B)*(u+v) = 2*A*B  ∧  x = A mcd B}
 {R3: x>0 ∧ y>0 ∧ u+v = 2*A*B/(A mcd B) = 2*(A mcm B) ∧  x = A mcd B }

 y:= (u+v)/2

 {Pos R: x = A mcd B  ∧  y = A mcm B}
]
```

[20/100]

2 *El k-simo elemento*

Considere un arreglo `b[0..n-1]` de números enteros, y un índice `k`, $0 \le k < n$. Se quiere desarrollar un algoritmo para encontrar el `k`–simo elemento más pequeño del arreglo (si `k=0`, la respuesta es el menor elemento; si `k=n-1`, la respuesta es el mayor elemento).

Estime el orden de complejidad temporal de este algoritmo, si la asignación es la operación básica.

Idea:

Permutar los elementos del arreglo con respecto a uno de sus elementos, i.e., un *pivote*, de modo que se pueda afirmar que los primeros elementos son menores o iguales al pivote y que los últimos son mayores al pivote. Suponga que el primer segmento tiene `p` elementos y el segundo `n-p`.

Tomar una decisión, después de comparar `k` con `p` y recurrir.

```
proc partir (var b[0..n-1],r,s, var j)
⌈
 {Pre QP: b=B ∧ r≤s}

 x,j,j1:= b[r],r,s;

 {Inv PP: perm(b,B)  ∧ b[0..r-1] = B[0..r-1]
                     ∧ b[r..j] ≤ x ∧ x < b[j1+1..s]
                     ∧ b[s+1..n-1] = B[s+1..n-1] }
 do j≠j1 →     if b[j] ≤ x →     j:= j+1
               [] b[j] > x →     b[j],b[j1]:= b[j1],b[j];
                                 j1:= j1-1
               fi
 od

 {Pos RP: perm(b,B)  ∧ b[0..r-1] = B[0..r-1]
                     ∧ b[r..j] ≤ b[j] < b[j+1..s]
                     ∧ b[s+1..n-1] = B[s+1..n-1] }
⌋
```

El predicado `perm(b,B)` es verdadero si `b` tiene los mismos valores de `B`, aunque pueden estar permutados.

Ahora:

```
[Ctx C: b[0..n-1]:int  ∧ 0≤k<n
 {Pre Q: b = B}

 partir(b,0,n-1,j)

 {Q1: perm(b,B)∧ b[0..j] ≤ b[j] < b[j+1..n-1]}

 do j≠k  →     if j<k      →     partir(b,0,j,j2);
                                   j:= j2
               [] j>k      →     partir(b,j,n-1,j2);
                                   j:= j2
               fi
 od
 {Pos R: perm(b,B) ∧ b[0..k-1] ≤ b[k] < b[k+1..n-1]}
```

El procedimiento `partir` ejecuta un ciclo que puede demorar `O(n)`.

El ciclo del programa entra, en el peor caso, `O(n)` veces, porque `partir` puede dejar uno de los subarreglos de la partición de tamaño `O(n)`.

En el peor caso: $T(n) = O(n^2)$.

[20/100]

3 Semianillos

Determine si las siguientes estructuras algebraicas son semianillos:

3a $(\mathbf{R}^*, \max, \min, 0, \infty)$

Sí es semianillo.

(**R***,max,0) es monoide conmutativo:

(1) x max (y max z) = (x max y) max z
(2) x max 0 = x
(3) x max y = y max x

(**R***,min,∞) es monoide

(4) x min (y min z) = (x min y) min z
(5) x min ∞ = x = ∞ min x

min distribuye sobre max

(6) x min (y max z) = (x min y) max (x min z)
(7) (y max z) min x = (y min x) max (z min x)

0 es anulador para min:

(8) 0 min x = x min 0 = 0

3b (**B**,≡,∨,true,false)

Sí es semianillo.

(**B**,≡,true) es monoide conmutativo:

(1) x ≡ (y ≡ z) ≡ (x ≡ y) ≡ z
(2) x ≡ true ≡ x
(3) x ≡ y ≡ y ≡ x

(**B**,∨,false) es monoide

(4) x ∨ (y ∨ z) ≡ (x ∨ y) ∨ z
(5) x ∨ false ≡ x
    false ∨ x ≡ x

∨ distribuye sobre ≡

(6) x ∨ (y ≡ z) ≡ (x ∨ y) ≡ (x ∨ z)
(7) (y ≡ z) ∨ x ≡ (y ∨ x) ∨ (z ∨ x)

true es anulador para ∨: (8) true ∨ x = x ∨ true = true

[20/100]

4 ***Misioneros y caníbales***

Tres misioneros y tres caníbales están en la margen derecha de un río y quieren cruzar a la margen izquierda. Tienen un bote en el que caben sólo dos personas. Si los caníbales sobrepasan en número a los misioneros, éstos serán devorados por los primeros.
Modele el problema como una búsqueda en grafos.

Estados: 0..3 × 0..3

```
V = {(mi,ci,b,md,cd)|
            mi: No. de misioneros en la margen izquierda,
            ci: No. de caníbales en la margen izquierda,
            b : bote a la derecha (=0) o a la izquierda (=1)
            md: No. de misioneros en la margen derecha,
            cd: No. de caníbales en la margen derecha}
```

Estado inicial: (3,3,0,0,0)

Arcos:

(mi,ci,b,md,cd) → (mi-2,ci,1-b,md+2,cd) , si mi-2≥ci, md+2≥cd
(mi,ci,b,md,cd) → (mi-1,ci,1-b,md+1,cd) , si mi-1≥ci, md+1≥cd
(mi,ci,b,md,cd)→ (mi-1,ci-1,1-b,md+1,cd+1) , si mi-1≥ci-1, md+1≥cd+1
(mi,ci,b,md,cd)→ (mi,ci-2,1-b,md,cd+2) , si mi≥ci-2, md≥cd+2

Problema: Buscar una ruta (3,3,0,0,0) →* (0,0,1,3,3).

[20/100]

5 *Devolución de cambio*

Suponga que se dispone de monedas con denominaciones $d_1, \ldots, d_n$ y que se quiere dar una cantidad de cambio A, usando la menor cantidad posible de monedas.
Desarrolle un algoritmo de programación dinámica para resolver el problema y estime su complejidad temporal y espacial en términos de A y de n.

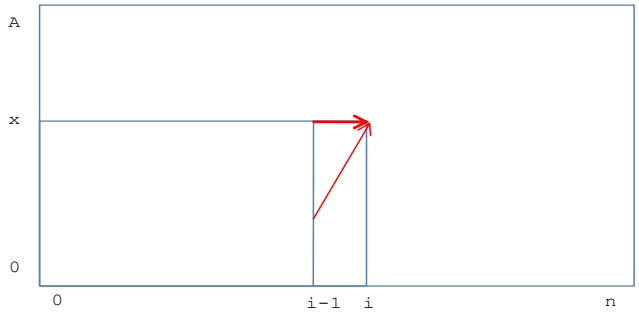
Lenguaje:

mnm(x,i) ≈ "mínimo No. de monedas para representar x, con las denominaciones $d_1, \ldots, d_i$ "
$0 \le x \le A,\ 0 \le i \le n$
mnm(A,n) = ?

Recurrencia:

mnm(x,i) = 0 , si $0=x,\ 0 \le i \le n$
= 0 , si $0 \le x \le A,\ 0=i$
= mnm(x,i-1) , si $0 \le x < d_i \le A,\ 0 < i \le n$
= mnm(x,i-1) min (1+mnm(x-d_i,i) , si $0 \le d_i \le x \le A,\ 0 < i \le n$

Diagrama de necesidades



Estructura de datos e Invariante

El esquema tiene la forma del esquema del problema del morral.

Basta un arreglo columna MNM[0..A]

El invariante:

Inv: MNM[0..x]=mnm(i-1,.) ∧ MNM[x+1..A]=mnm(i,.) ∧ $0 < i \le n+1$

Complejidades

S(A,n) = θ(A)

$T(A,n) = \theta(An)$

6 *Babel Ltda.*

Babel Ltda. ha desarrollado un novedoso método de construcción de torres que le permite alcanzar grandes altitudes sin mayores dificultades. Su diseño se basa en la disponibilidad de placas prefabricadas cuadradas de 100 m de lado y 1 m de altura. Construir una de tales placas lleva una semana, pero Babel Ltda. tiene proveedores que le pueden suministrar, sin demoras adicionales, todas las placas que necesite para la construcción.

El método consiste, simplemente, en apilar placas. Para esto, la empresa usa unos gatos que pueden colocar una pila de placas sobre otra, en una semana de trabajo. Si la altura de alguna de las dos pilas involucradas en este proceso supera 100 m, el trabajo exige una semana adicional.

Babel Ltda. quiere evaluar el tiempo mínimo, en semanas, que puede emplear para la construcción de una torre de N metros de altura.

6a Utilice programación dinámica para estimar el tiempo mínimo de construcción de una torre de altura h, $1 \le h \le N$. (definición de lenguaje, recurrencia, diagrama de necesidades, invariante). No es necesario que escriba su algoritmo.

Lenguaje

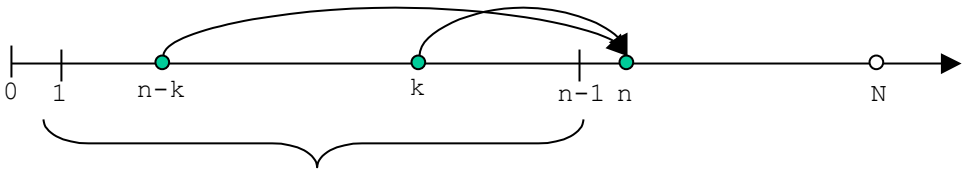
$tmin(h) \approx$ "No. mínimo de semanas para construir una torre de h metros"

$tmin(N) = ?$

Recurrencia

$$\begin{aligned} tmin(h) &= 1 && , \text{ si } h=1 \\ &= (\min k \mid 0<k<h : \max(tmin(k), tmin(h-k)) + 1 \\ &\qquad + \textbf{if } k>100 \vee h-k>100 \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi}) && , \text{ si } h>1 \end{aligned}$$

Diagrama de necesidades



Invariante

$Inv: 0<h \le N \wedge (\forall i \mid 0<i<h : TMIN[i] = tmin(i))$

Variante

Sea $f(k) = \max(tmin(k), tmin(h-k)) + 1 + \textbf{if } k>100 \vee h-k>100 \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi}$.

Así, para $h>1$: $tmin(h) = (\min k \mid 0<k<h : f(k))$.

Es fácilmente comprobable, debido a que $f(k) = f(h-k)$:

Lema A: Para $h>1$:

```
    tmin(h) = (min k | 0 < k ≤ ⌈h/2⌉ : max(tmin(k),tmin(h-k)) + 1
                                    + if k>100 ∨ h-k>100 then 1 else 0 fi)
```

∎

El siguiente resultado es previsible, aunque algo engorroso de mostrar:

Lema B: `tmin` es creciente

Demostración: Es claro que $tmin(2)=2>1=tmin(1)$. Supóngase que el resultado vale para $h>1$. Ahora:

```
   tmin(h+1)
=
   (min k | 0<k<h+1 : max(tmin(k),tmin(h+1-k)) + 1
                                    + if k>100 ∨ h+1-k>100 then 1 else 0 fi)
=     ⟨partir rango⟩
   (min k | 0<k<h : max(tmin(k),tmin(h+1-k)) + 1
                                    + if k>100 ∨ h+1-k>100 then 1 else 0 fi)
      min (max(tmin(h),tmin(h+1-h)) + 1 + if h>100 ∨ h+1-h>100 then 1 else 0 fi)
   =
   (min k | 0<k<h : max(tmin(k),tmin(h+1-k)) + 1
                                    + if k>100 ∨ h+1-k>100 then 1 else 0 fi)
      min (max(tmin(h),tmin(1)) + 1 + if h>100 then 1 else 0 fi)
≥     ⟨tmin(h) ≥ tmin(1)⟩
   (min k | 0<k<h : max(tmin(k),tmin(h+1-k)) + 1
                                    + if k>100 ∨ h+1-k>100 then 1 else 0 fi)
      min (tmin(h) + 1 + if h>100 then 1 else 0 fi)
≥     ⟨H.I.: tmin(h+1-k)>tmin(h-k). Además: h-k>100 ⇒ h+1-k>100⟩
   (min k | 0<k<h : max(tmin(k),tmin(h-k)) + 1
                                    + if k>100 ∨ h-k>100 then 1 else 0 fi)
   min (tmin(h) + 1 + if h>100 then 1 else 0 fi)
=
   tmin(h) min (tmin(h) + 1 + if h>100 then 1 else 0 fi)
=
   tmin(h)
```

∎

En estas condiciones:

```
    tmin(h)
=
    (min k | 0 < k ≤ ⌈h/2⌉ : max(tmin(k),tmin(h-k)) + 1
                                    + if k>100 ∨ h-k>100 then 1 else 0 fi)
=
    (min k | 0 < k ≤ ⌈h/2⌉ : max(tmin(k),tmin(h-k)) + 1
                                    + if k>100 ∨ h-k>100 then 1 else 0 fi)
=
    (min k | 0 < k < ⌈h/2⌉ : tmin(h-k) + 1
                                    + if k>100 ∨ h-k>100 then 1 else 0 fi)
   min (max(tmin(⌈h/2⌉),tmin(h-⌈h/2⌉)) + 1
                                    + if ⌈h/2⌉>100 ∨ h-⌈h/2⌉>100 then 1 else 0 fi)
=
    (min k | 0 < k < ⌈h/2⌉ : tmin(h-k) + 1
                                    + if k>100 ∨ h-k>100 then 1 else 0 fi)
```

```
   min (tmin(⌈h/2⌉) + 1 + if ⌈h/2⌉>100 then 1 else 0 fi)
=
   tmin(⌈h/2⌉) + 1 + if ⌈h/2⌉>100 then 1 else 0 fi
```

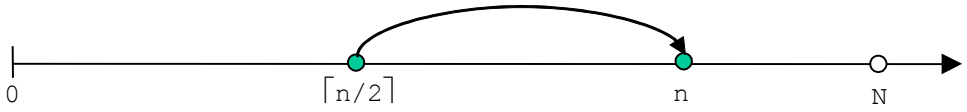
Es decir, se puede plantear la recurrencia más simple:

Recurrencia:

```
   tmin h    = 1                                                    , si h=1
             = tmin(⌈h/2⌉) + 1 + if ⌈h/2⌉>100 then 1 else 0 fi      , si h>1
```

Explicación de la corrrección de la recurrencia (no necesariamente formal!)

Diagrama:



Invariante:

Inv: $0<h\leq N \wedge (\forall i \mid 0<i<h : TMIN[i] = tmin(i))$

6b Estime las complejidades espacial y temporal de su algoritmo en términos de N. Para lo temporal, use la asignación como operación básica.

Complejidad espacial:

Se usa el arreglo $TMIN[1..N]$. Entonces:

$S(N) = \theta(N)$

Complejidad temporal:

El cálculo de $tmin(h)$ exige la evaluación de las expresiones

```
f(k) = max(tmin(k),tmin(h-k)) + 1 + if k>100 ∨ h-k>100 then 1 else 0 fi
```

para $0<k<h$. Es decir, la iteración h del invariante conlleva $\theta(h-1)$ cálculos. Así:

$$
\begin{aligned}
T(N) &= (+h \mid 0<h\leq N : \theta(h-1)) \\
&= \theta(+h \mid 0<h\leq N : h-1) \\
&= \theta(N(N-1)/2) \\
&= \theta(N^2)
\end{aligned}
$$

Si se observa que $f(k) = f(h-k)$, para $0<k<h$, es posible reducir el número de cálculos por iteración a sólo la mitad. En otras palabras:

$$
\begin{aligned}
T(N) &= \theta(N(N-1)/4) \\
&= \theta(N^2)
\end{aligned}
$$

VARIANTE

Complejidad espacial:

Se usa el arreglo `TMIN[1..N]`. Entonces:
$S(N) = \theta(N)$

Complejidad temporal:

La expresión

$$tmin(\lceil h/2 \rceil) + 1 + \textbf{if } \lceil h/2 \rceil > 100 \textbf{ then } 1 \textbf{ else } 0 \textbf{ fi}$$

es calculable en $O(1)$. De esta manera:

$$T(N) = N\,\theta(1) = \theta(N)$$

[20/100]