ISIS 211 Diseño de algoritmos
Semestre 2002-2 * Parcial No. 1
Septiembre 26, 2002
Prof. Rodrigo Cardoso

---

1 **[20 puntos]** Enriquezca GCL con una instrucción `FOR`, de la forma

```
for (INIC ; B ; INCR) S rof
```

con semántica operacional como en C, i.e.,

i. Se ejecuta la instrucción `INIC`.
ii. Si el predicado `B` no vale, se termina la ejecución. En otro caso se continúa.
iii. Se ejecuta la instrucción `S`.
iv. Se ejecuta la instrucción `INCR`.
v. Se va al paso ii.

1a [10/20] Simule `FOR` con un programa `GCL` que no utilice esta instrucción.

1b [10/20] Enuncie un "Teorema del FOR" que permita concluir la corrección de afirmaciones como

```
{Q}  for (INIC ; B ; INCR) S rof {R}
```

1c *[opcional: 10/20]* Use el resultado de **b** para mostrar que

```
                {Q : n>0}
for (i,p:= 1,1 ; i≠n+1 ; i:= i+1) p:= p*i rof
               {R : p = n!}
```

---

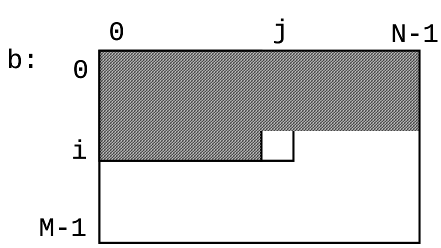
2 **[50 puntos]**

2a [40/50] Un arreglo `b[0..M-1,0..N-1]` **of bool** tiene los resultados de `N` juegos de béisbol de `M` equipos, de modo que `b[i,j]` ≡ "equipo `i` ganó el juego `j`" para `(i,j) ∈ 0..M-1 × 0..N-1`. Una *racha ganadora* del equipo `i` es una sucesión de juegos que `i` haya ganado en forma consecutiva. Una *racha perdedora* es una sucesiuón de juegos perdidos en forma consecutiva.
Problema: Se quiere especificar y desarrollar un programa para determinar los tamaños de las rachas ganadoras y perdedoras más largas. Complete los ítems que faltan ("?")en el siguiente desarrollo:

```
[Ctx C: b[0..M-1,0..N-1] of bool

    ?

    {Inv P: 0 ≤ i ≤ M  ∧ 0 ≤ j < N ∧
```




```
        g = rgml(b: ■ )  ∧  g1 = rgml1(b: ■ ) ∧
        p = rpml(b: ■ )  ∧  p1 = rpml1(b: ■ )
    }
    {cota t: ? }

    do ? →  ? od
    {Pos R: g = rgml(b) ∧ p = rpml(b)}
]
```

donde:

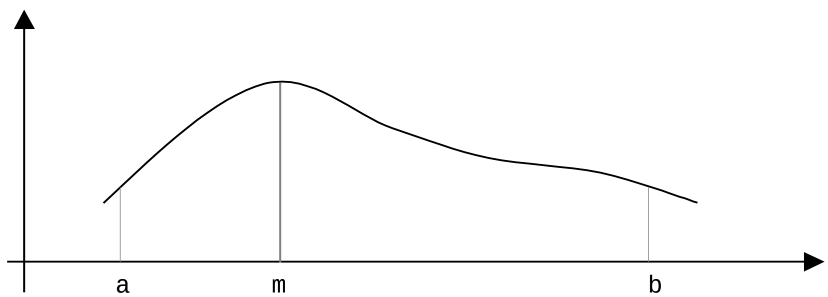
```
rgml(xxx)  ≈  "tamaño de racha ganadora más larga en xxx"
rgml1(xxx) ≈  "tamaño de racha ganadora más larga en xxx, y que termine en
               el último elemento considerado"
rpml(xxx)  ≈  "tamaño de racha perdedora más larga en xxx"
rpml1(xxx) ≈  "tamaño de racha perdedora más larga en xxx, y que termine en
               el último elemento considerado"
```

2b [10/50] Estime el orden de la complejidad de su algoritmo, tomando como operación básica la asignación (simple o paralela) de variables.

---

3 [30/30]

Una función `f:[a,b]` → **`real`** es *unimodal* si es continua y existe un punto `m`, con a≤m≤b, tal que `f` es creciente en `[a,m]` y decreciente en `[m,b]`.



El predicado `unimodal(f,a,m,b)` denotará esta situación. Dado un número real `eps>0`, considere el siguiente un programa para determinar un valor de `x`∈`[a,b]` que aproxime a `m` con un error menor que `eps`.

```
[Ctx C: unimodal(f,a,m,b) ∧ eps>0

     p,q:= a,b;

     {Inv P : a ≤ p ≤ q ≤ b ∧ unimodal(f,p,m,q) ∧ x = (p+q)/2 }
     {cota t: q-p-eps }

     do q-p ≥ eps →     u,v:= p+(q-p)/3,q-(q-p)/3;
                         if f(u)≤f(v) → p:= u
                         [] f(u)>f(v) → q:= v
                         fi;
                         x:= (p+q)/2
     od

     {Pos R: |x-m| < eps}
]
```

3a [10/30] Demuestre que el invariante sirve para mostrar la poscondición.

3b [20/30] Estime la complejidad del algoritmo.

ISIS 211 Diseño de algoritmos
Semestre 2002-2 * Solución Parcial No. 1
Septiembre 26, 2002
Prof. Rodrigo Cardoso

---

1 [20/20]

1a [10/20]

for (INIC ; B ; INCR) S **rof** $\cong$ INIC ; **do** B $\rightarrow$ S ;INCR **od**

```
INIC;                    [3/10]
do B                     [3/10]
    →   S;               [4/10]
        INCR
od
```

1b [10/20]

Teorema del FOR

Con las definiciones de 1a:

Si existe un predicado P tal que : [2/10]

[a] {Q} INIC {P} [2/10]

[b] P $\wedge$ ¬B $\Rightarrow$ R [2/10]

[c] {P $\wedge$ B} S ; INCR {P} [2/10]

[d] Hay terminación (v.gr., se pueden dar condiciones sobre una función cota). [2/10]

Entonces:

{Q } **for** (INIC ; B ; INCR) S **rof** {R}

1c [Opcional: 10 puntos]

En este caso : [2/10]

INIC $\cong$ i,p:= 1,1

B $\equiv$ i≠n+1

INCR $\cong$ i:= i+1

S $\cong$ p:= p*i

Sea P $\equiv$ 1≤i≤n+1 $\wedge$ p = (i-1)!

Para verificar las condiciones del teorema, se debería mostrar:

[a] {n>0} i,p:= 1,1 {P} [2/10]

[b] 1≤i≤n+1 $\wedge$ p = (i-1)! $\wedge$ i=n+1 $\Rightarrow$ p = n! [2/10]

[c] {1≤i≤n+1 $\wedge$ p = (i-1)! $\wedge$ i≠n+1} p:= p*i ; i:=i+1 {P} [2/10]

[d] Terminación. [2/10]

---

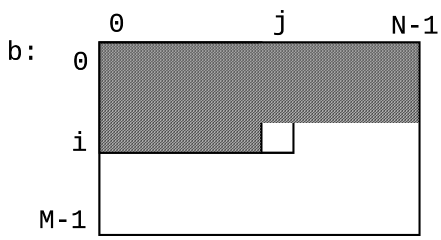
2 **[50/50]**

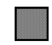
2a [40/50]

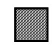
[Ctx C: b[0..M-1,0..N-1] **of bool**

i,j,g,g1,p,p1:= 0,0,0,0,0,0; [5/40]

{Inv P: 0 ≤ i ≤ M ∧ 0 ≤ j < N ∧



g = rgml(b: ) ∧ g1 = rgml1(b: ) ∧

p = rpml(b: ) ∧ p1 = rpml1(b: )

}

{cota t: (M-i)*N-j} [5/40]

```
do i≠M →                                            [5/40]
        if b[i,j] → g1,p1:= g1+1,0                  [8/40]
        []¬b[i,j] → g1,p1:= 0,p1+1
        fi;
        if g1>g → g:= g1                            [5/40]
        [] g1≤g → skip
        fi;
        if p1>p → p:= p1                            [5/40]
        [] p1≤p → skip
        fi;
        if j≠N-1 → j:= j+1                          [7/40]
        [] j=N-1 → i,j,g1,p1:= i+1,0,0,0
        fi
od
{Pos R: g = rgml(b) ∧ p = rpml(b)}
```

]

2b [10/50]

El cuerpo del ciclo es O(1). Las iteraciones se repiten tantas veces como elementos en la matriz. Es decir, el algoritmo es O(MN). [10/10]

---

3 **[30/30]**

3a [10/30] Demuestre que el invariante sirve para mostrar la poscondición.

Hay que mostrar que el invariante y la negación de la guarda implican la poscondición, i.e.,

```
a ≤ p ≤ q ≤ b ∧ unimodal(f,p,m,q) ∧ x = (p+q)/2 ∧ q-p < eps ⇒ |x-m| < eps
```

Variante 1

```
Lema A: p ≤ q  ∧  x = (p+q)/2 ⇒  p ≤ x ≤ q
Dem:
     Hip: p ≤ q , x = (p+q)/2
     p ≤ x ≤ q
=         ⟨x = (p+q)/2⟩
     p ≤ (p+q)/2 ≤ q
=
     2*p ≤ p+q ≤ 2*q
=
     2*p ≤ p+q  ∧  p+q ≤ 2*q
=
     p ≤ q
=         ⟨Hip: p ≤ q⟩
     true

Lema B: p ≤ q  ∧ unimodal(f,p,m,q) ⇒  p ≤ m ≤ q
Dem:
     Hip: p ≤ q , unimodal(f,p,m,q)
     unimodal(f,p,m,q)
⇒         ⟨Def unimodal⟩
     p ≤ m ≤ q


Dem (P ∧ ¬BB ⇒ R)

Hip: a ≤ p ≤ q ≤ b, unimodal(f,p,m,q), x = (p+q)/2, q-p < eps
     true
=          ⟨Lema A, Lema B⟩
     p ≤ x ≤ q  ∧  p ≤ m ≤ q
=
     p ≤ x ≤ q  ∧  -q ≤ -m ≤ -p
⇒
     p-q ≤ x-m ≤ q-p
⇒         ⟨Hip: q-p < eps, monotonía⟩
     -eps < x-m < eps
=
     |x-m| < eps
```

Variante 2

`x` y `m` están en el intervalo `[p,q]`. Si la distancia entre los extremos (`q-p`) es menosr que `eps`, la distancia entre dos puntos del intervalos debe ser menor que `eps`.

3b [20/30]

Estime la complejidad del algoritmo.

En cada iteración el intervalo se reduce a 2/3 del valor en que comenzó la iteración.

Sea L = b-a, la longitud del intervalo inicial. Después de k iteraciones, la longitud del intervalo [p,q] es L*(2/3)^k.

Al terminar, digamos en la iteración n, éste es el primer valor para el que:

$$L*(2/3)^n < eps$$

=

$$\log L + n \log(2/3) < \log eps$$

=

$$n > \frac{\log L/eps}{\log 3/2}$$

i.e. la complejidad temporal es O(log (b-a)/eps).