**ISIS 2103 Diseño de algoritmos**
**Semestre 2011-10 Parcial 2**
**Abril 12, 2011**
**Prof. Rodrigo Cardoso**

**1** **(30/100)** Los resultados históricos de los puntos ganados por un equipo de fútbol están guardados en un arreglo `b:[0..n-1] of {0,1,3}`. En el partido `i`, `0≤i<n`, `b[i]`es `0`, `1` ó `3`, según que el equipo haya perdido, empatado o ganado, respectivamente.
Se quiere determinar la longitud de la *racha no perdedora más larga* (máximo número de partidos consecutivos, sin perder), en la historia del equipo.
Use la siguiente notación, para `0≤i<n`:
`rnpml(i)` ≈ "longitud de la racha no perdedora más larga en `b[0..i-1]`"
`rnpmla(i)` ≈ "longitud de la racha no perdedora más larga en `b[0..i-1]`, terminando en `b[i-1]`"

Considere un algoritmo que responda a una especificación de la forma:

```
[Ctx C: n: nat ∧ b:[0..n-1] of {0,1,3}
 ...
 {Inv P}
 {cota t }
 do  ...  od
 {R : r = rnpml(n) }
]
```

**1a (10/30)** Defina un invariante `P1` a partir de la especificación, usando la técnica de cambiar una constante por una variable. Defina `t` correspondiente a su definición de `P1`.

Se cambia en `R` la variable `n` por `i`, para variar en el rango `0≤i≤n`:
`P1 ≡ 0≤i≤n ∧ r=rnpml(i)`

**[5/10]**

La cota es la distancia de `i` a `n`:
`t = n-i`

**[5/10]**

**1b (10/30)** Defina un invariante `P`, que fortalezca `P1`, de modo que la solución que se desarrolle sea lineal en `n`, contando asignaciones como operaciones básicas.

Se define `P` como `P1` más el compromiso de mantener la longitud de la más grande racha sin perder que termina en `b[i-1]`:
`P ≡ 0≤i≤n ∧ r=rnpml(i) ∧ ra=rnpmla(i).`

**[10/10]**

Claramente: `P` ⇒ `P1`, i.e., `P` es un fortalecimiento de `P1`.

**1c [10/30]** Escriba el código correspondiente al invariante `P` y a la cota `t` establecidas.

```
[Ctx C: n: nat ∧ b:[0..n-1] of {0,1,3}

 i:= 0;
 if b[0]=0   then r,ra:= 0,0
             else r,ra:= 1,1
 fi;
```

```
{Inv P : 0≤i≤n ∧ r=rnpml(i) ∧ ra=rnpmla(i)}
{cota t: n-i }

do  i≠n →  if b[i]=0   then ra:= 0
                       else ra:= ra+1
           fi;
           if ra>r     then r:= ra
                       else skip
           fi;
           i:= i+1
od

{R : r = rnpml(n) }
]
```

**[10/10]**

**2** **(30/100)** La conjetura de Goldbach establece que todo número par, mayor que 2, puede expresarse como la suma de dos números primos (v.gr., `12 = 5+7, 14 = 7+7`). No es un teorema demostrado, pero se ha comprobado experimentalmente para todos los pares menores a $10^{18}$.
Suponga conocido un arreglo `p[1..k]` que guarda ordenados los primeros k primos impares, de manera que `p[k]`$\leq 10^{18}$.
Dado un número par n, 2<n≤`p[k]`, se quieren determinar dos números primos cuya suma sea n.
**N.B.** Documente su programa con aserciones, explicaciones de paradigmas, etc. *Se calificarán mejor las soluciones más eficientes.*

**2a (20/30)** Escriba una solución para el problema

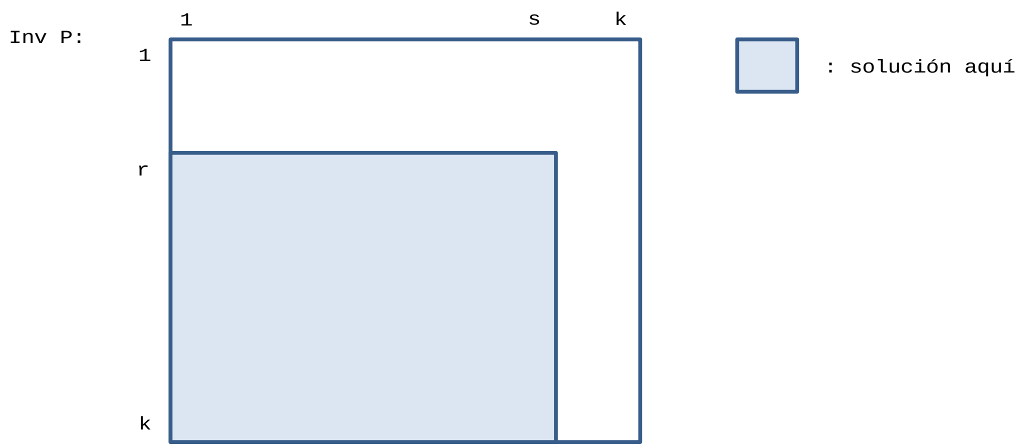
```
[Ctx C: n:nat ∧ par.n ∧ 2<n≤p[k]≤10^18 ∧ "p:[1..k]: primeros k primos"
 {Pos R: n = p[r]+p[s]}
]
```

***Variante 2a1***

Se plantea una búsqueda con certeza en la que el invariante refleja "apretar el cerco". Nótese que la función

`f: 1..k ×1..k → nat`

definida por `f(i,j)=p[i]+p[j]` es creciente en cada componente. El invariante que se propone es de la forma




que se puede formalizar así:

```
P ≡ (∃i,j| r≤i≤k ∧ 1≤j≤s: p[i]+p[j]=n) ∧ 1≤r≤k ∧ 1≤s≤k
```

La cota t puede medir el área de incertidumbre:

```
t = (k-r+1)*s     , si p[r]+p[s]≠n
    0             , si p[r]+p[s]=n
```

o el número de filas y columnas de esta área, i.e.,

```
t = k-r+1+s , si p[r]+p[s]≠n
    0       , si p[r]+p[s]=n
```

Así, el algoritmo solución será (se usa la segunda definición para `t`):

```
[  Ctx C: n:nat ∧ par.n ∧ 2<n≤p[k]≤10^18 ∧ "p:[1..k]: primeros k primos"

   r:= 1;
   s:= k;

   {Inv P : (∃i,j| r≤i≤k ∧ 1≤j≤s: p[i]+p[j]=n) ∧ 1≤r≤k ∧ 1≤s≤k }
   {cota t: if p[r]+p[s]≠n then k-r+1+s else 0 fi}
```

```
   do  p[r]+p[s]>n      →     r:= r+1
   []  p[r]+p[s]<n      →     s:= s-1
   od

   {R : n = p[r]+p[s] }
]
```

**Documentación y explicaciones [10/20]**
**Algoritmo [10/20]**
**Bono de eficiencia [+10]**

### *Variante 2a2*

Se puede hacer una búsqueda lineal con certeza en el espacio `1..k ×1..k`:

```
E = 1..k ×1..k                // Espacio de búsqueda
suc(i,j) = (i,j+1), si j≠k    // Sucesor
         = (i+1,1), si j=k
e0 = (1,1)                    // Primer elemento
q(i,j) ≡ p[i]+q[j]=n          // Predicado de búsqueda
```

La solución propuesta es:

```
[  Ctx C: n:nat ∧ par.n ∧ 2<n<10^18 ∧ "p:[1..k]: primeros k primos"

   r,s:= 1,1;
   do p[r]+q[s]≠n →     if s≠k      then s:= s+1
                                    else r,s:= r+1,1
                        fi
   od
   {Pos R: n = p[r]+p[s] }
]
```

**Documentación y explicaciones [10/20]**
**Algoritmo [10/20]**

### *Variante 2a3*

Hacer ciclos anidados de búsquedas con incertidumbre (así se tenga seguridad de que se hallará una pareja de primos que sumen `n`):

```
[  Ctx C: n:nat ∧ par.n ∧ 2<n<10^18 ∧ "p:[1..k]: primeros k primos"

   r,rcent:= 1,k;
   do r≠rcent →  s,scent:= 1,k;
                 do s≠scent → if p[r]+q[s]≠n then s:= s+1
                                              else rcent,scent:= r,s
                              fi
                 od;
                 if r≠rcent  then r:= r+1
                             else skip
                 fi
   od
   {Pos R: n = p[r]+p[s] }
]
```

**Documentación y explicaciones [10/20]**
**Algoritmo [10/20]**

**2b (10/30)** Estime las complejidades espacial y temporal de su solución. Operación básica: comparación.

***Variante 2a1***

Complejidad espacial: solo hay 2 variables, `r` y `s`

`S(n) = θ(1).`

**[5/10]**

Complejidad temporal: Ciclos anidados, cada uno de tamaño `k`. En cada iteración se cambia en `1` una de las variables. El peor caso es cuando `r` cambia de `1` a `k` y `s` cambia de `k` a `1`. Habría `2k` pasos.

`T(n) = θ(k).`

**[5/10]**

***Variante 2a2***

Complejidad espacial: solo hay 2 variables, `r` y `s`

`S(n) = θ(1).`

**[5/10]**

Complejidad temporal: El espacio de búsqueda mide $k^2$. Es posible que deba recorrerse en su totalidad.

$T(n) = \theta(k^2).$

**[5/10]**

***Variante 2a3***

Complejidad espacial: solo hay 4 variables, `r, rcent, s, scent`.

`S(n) = θ(1).`

**[5/10]**

Complejidad temporal: Ciclos anidados, cada uno de tamaño `k`.

$T(n) = \theta(k^2).$

**[5/10]**

**3** **(40/100)** Juan ha decidido planear su ahorro y sus gastos diarios. El día `0`, Juan ahorra $`1` y no gasta nada, por lo que tiene un total de $`1`. De allí en adelante, el día `d`, si `d` es par, ahorra el doble de lo que ahorró en el día `d/2` y gasta lo que gastó en el día `d/2`. En cambio, si `d` es impar, ahorra lo que ahorró el día anterior y gasta todo lo que tenía el día anterior, menos $`1`.

**3a (30/40)** Diseñe un algoritmo de programación dinámica para calcular el total de dinero que Juan posee, en el día `D`. Use como lenguaje del problema la siguiente notación:
`a(d)` ≈ "ahorro en el día d"
`g(d)` ≈ "gasto en el día d"
`t(d)` ≈ "total en el día d".

*Lenguaje*
`a(d)` ≈ "ahorro en el día d"
`g(d)` ≈ "gasto en el día d"
`t(d)` ≈ "total en el día d"

```
t(D) = ?
```

*Recurrencia*
```
a(0) = 1
a(d) = 2*a(d/2)    , si 0<d, par(d)
     = a(d-1)      , si 0<d, ¬par(d)
```

**[5/30]**

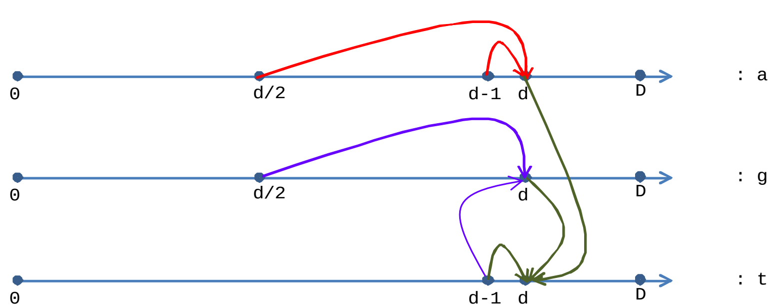
```
g(0) = 0
g(d) = g(d/2)      , si 0<d, par(d)
     = t(d-1)-1    , si 0<d, ¬par(d)
```

**[5/30]**

```
t(0) = 1
t(d) = t(d-1)+a(d)-g(d) , si 0<d
```

**[5/30]**

*Diagrama de necesidades*




**[5/30]**

*Estructura de datos + Invariante*

Se pueden usar 3 estructuras de datos
`A[0..D]` **`of nat`** `:` para guardar valores de ahorro

`G[0..D]` **`of nat`** `:` para guardar valores de gasto
`T` `:` para guardar el total

**[5/30]**

El invariante será:
`Inv P: 0≤d≤D ∧ (∀i| 0≤i≤d : A[i]=a(i) ∧ G[i]=g(i)) ∧ T=t(d)`

**[5/30]**

**3b (10/40)** Estime las complejidades espacial y temporal de su solución. Operación básica: asignación.

Complejidad espacial: dos arreglos de tamaño `D` más una variable.
`S(D) = θ(D).`

**[5/10]**

Complejidad temporal: calcular `A[d],G[d],T` para `d=0,1,…,D`. Cada iteración cuesta `θ(1)`, y son `D` iteraciones.
`T(D) = θ(D).`

**[5/10]**

El algoritmo (no cuenta el escribirlo, solo por ilustración)

```
[Ctx : …

 A[0],G[0],T:= 1,0,1;
 d:= 1;

 {Inv P: 0≤d≤D ∧ (∀i| 0≤i≤d : A[i]=a(i) ∧ G[i]=g(i)) ∧ T=t(d)}
 do d≠D  → if par(d)   then  a[d],g[d]:= 2*a[d/2],g[d/2]
                       else  a[d],g[d]:= a[d-1],T-1
           fi;
           T:= T+a[d]-g[d];
           d:= d+1
 od

 {Pos: T=t(D)}
]
```

**[0/10]**