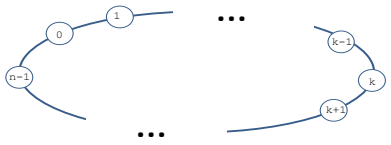
ISIS 1105 Diseño y Análisis de Algoritmos **Nombre**: _____
Semestre 2010-20 – Parcial 2
Octubre 26, 2010
Prof. Rodrigo Cardoso

En el contexto de este examen:

- *Si se pide describir algoritmo se debe entender que se califican mejor los algoritmos más eficientes. Una descripción no obliga a escribir formalmente el algoritmo, sino a explicar lo que hace, de manera -posiblemente- informal.*
- *Si se pide estimar una complejidad algorítmica, debe justificarse la respuesta.*

1 [40/100]

Una instalación de luces navideñas consta de n bombillas, denotadas $0, 1, \ldots, n-1$, y conectadas en un anillo de la forma:



Las bombillas tienen un mecanismo que las enciende o las apaga, una vez por segundo, dependiendo del estado de sus vecinas. El tiempo se cuenta desde 0 y por segundos y, en el segundo 0, las bombillas tienen una configuración inicial de apagado o encendido dada.
Para el segundo $j>0$, una bombilla se enciende si y solo si, en el segundo $j-1$:
- la bombilla está encendida y las dos bombillas, anterior y siguiente en el anillo, están encendidas; o
- la bombilla está apagada y alguna de las dos bombillas, anterior y siguiente en el anillo, está encendida.

En caso de que exista, se quiere determinar, mediante un algoritmo de programación dinámica, el *tiempo de inicio de repetición* de la instalación, definido como el menor tiempo t tal, que la configuración de encendido de las bombillas en t se repita en el futuro.

Sea $b(i,j)$ el estado de la bombilla i en el segundo j, para $0 \le i < n$, $j \ge 0$.

1a Establezca una recurrencia que defina la función b.

Variante 1a1

```
b: 0..n-1 × nat  →  bool      // false: apagado; true: encendido
```

Sea b_{i0} el estado de la bombilla i al comenzar, $0 \le i < n$.

```
b(i,j)      = b_i0  , si 0≤i<n, 0=j
            = if b(i,j-1)
                  then b((i-1) mod n,j-1) ∧  b((i+1) mod n,j-1)

                  else b((i-1) mod n,j-1) ∨  b((i+1) mod n,j-1)
              fi
                  , si 0≤i<n, 0<j.
```

[10/40]

Variante 1a2

```
b: 0..n-1 × nat  →  {0,1}     // 0: apagado; 1: encendido
```

Sea b_{i0} el estado de la bombilla i al comenzar.

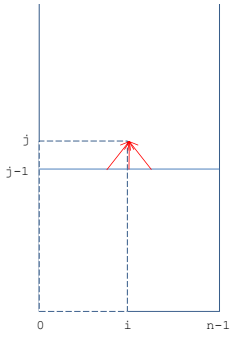
```
b(i,j)      = b_i0 , si 0≤i<n, 0=j
            = if b(i,j-1)=1
                 then b((i-1) mod n,j-1)* b((i+1) mod n,j-1)

                 else max(b((i-1) mod n,j-1),b((i+1) mod n,j-1))
              fi
                 , si 0≤i<n, 0<j.
```

[10/40]

1b Estime las complejidades espacial y temporal de un algoritmo de programación dinámica que haga uso de la recurrencia planteada en **1a** para decidir si el tiempo de inicio de repetición está bien definido y, en este caso, calcularlo.

Diagrama de necesidades



Las dependencias se entienden `mod n`. Es decir, en los extremos de una línea horizontal se requieren valores en la fila `j-1` que están al otro extremo.

[5/40]

Estructura de datos e invariante

Variante 1b1

Estructura de datos:
Dos arreglos para guardar el estado de luces en segundos consecutivos.
Tamaño de cada arreglo: `n` (un bit por cada bombilla)
Una lista para guardar, codificados, los estados por los que se ha pasado.
Tamaño de esta lista: 2^n (peor caso)
Los estados se deben repetir, porque las luces pueden considerarse como representación de un número binario de `n` bits. Hay 2^n posibles números.
Un estado `b(0..n-1,j)`, se codifica con el valor numérico correspondiente (un número natural):
$bin_j = (+i|0\leq i\leq n : b(i,j)*2^i)$
bin_j es único para representar un estado específico.

Así:

```
B0[0..n-1] of {0,1}
B[0..n-1] of {0,1}
C: seq of 0..2^n-1
```

[5/40]

Con el invariante:

$Inv: 0<j \wedge B[0..i-1]=b(0..i-1,j)$
$\wedge (\forall k| 0\leq k<i: C.k = bin_k) \wedge (\forall k,k1| 0\leq k<k1<i: C.k \neq C.k1)$

[10/40]

Complejidades

Espacio: $S(n) = \theta(2*n) + \theta(2^n)$
$= \theta(2^n)$

[5/40]

Tiempo:

En la iteración j se calcula $b(.,j)$ en $B[j]$. Para esto:

$B0 := B$: copia de B en B0
Total: $\theta(n)$.

Cálculo de b(i,j) : determinar nuevo B[i], a partir de B0
$O(1)$ por cada $b(i,j)$, $0 \le i < n$.
Total: $\theta(n)$

$C.j := bin_j$: cálculo de bin_j
Total: $\theta(n)$.

El peor caso tendrá que repetir este proceso 2^n veces antes de encontrar un repetido.

$T(n) = \theta(n*2^n)$

[5/40]

Posibles mejoras:

Mejora 1:

El espacio se puede reducir, usando solo 3 variables que guarden los valores que sirven para calcular cada $b(i,j)$:

```
{Inv}
i:= 0;
a0,b0,c0:= B[n-1],B[0],B[1];
{Inv': Inv ∧ 0≤i≤n ∧ B[0..i-1]=b(0..i-1,j) ∧ B[i..n-1]=b(i..n-1,j-1)
       ∧ a0=b((i-1) mod n,j-1) ∧ b0=b(i,j-1) ∧ c0=b((i+1) mod n,j-1)}
do i≠n →    if b0=1
                  then B[i]:= a0*c0
                  else B[i]:= max(a0,c0)
            fi;
            i:= i+1;
            a0,b0,c0:= b0,c0,B[i mod n];
od
{Q1: Inv ∧ B[0..n-1]= b(.,j)}
```

Bono: +5/40

Mejora 2

Experimentalmente, parece que el algoritmo converge mucho más rápido que 2^n, y que el período (lo que tarda en repetirse en patrón de luces) es siempre 2.

Se otorgará un bono a quien demuestre este resultado o algo similar que, de ser cierto aceleraría considerablemente los algoritmos mostrados.

Fecha Límite: Noviembre 2, 2010.

Bono: +10/40

2 **[30/100]**

Sea $G(V,E)$ un grafo dirigido, con $V = \{1,2,\ldots,n\}$. Dados $p,q,r \in V$, considere el problema de determinar si existe un circuito que pase por los nodos p, q y r.

2a Describa un algoritmo para solucionar el problema

El circuito puede ser $p \rightarrow^* q \rightarrow^* r \rightarrow^* p$, o bien $p \rightarrow^* r \rightarrow^* q \rightarrow^* p$.

G se puede representar con un arreglo de listas de adyacencias.

El algoritmo de Dijkstra se puede usar para resolver el problema de conectividad $x \rightarrow^* y$ para cualesquier par de nodos $x,y \in V$. El costo de una pregunta es $O(n \log n + e)$.

Ahora, se puede decidir si hay un ciclo $p \rightarrow^* q \rightarrow^* r \rightarrow^* p$ en $O(3*(n \log n + e))$, es decir, en $O(n \log n + e)$. Si esto falla, se debe buscar si hay un ciclo $p \rightarrow^* r \rightarrow^* q \rightarrow^* p$, con costo $O(n \log n + e)$.

En el peor caso hay que hacer 6 preguntas y, en total, son $O(n \log n + e)$.

[20/30]

Hacer 3 preguntas en lugar de 6.

[-5/30]

2b Estime las complejidades espacial y temporal de su algoritmo.

$S(n) = O(n)$ // Para guardar marcados

[5/30]

$T(n) = O(n \log n + e)$

[5/30]

3 **[30/100]**

Sea R una relación binaria sobre $V = 1..n$. Se dice R es *acíclica* cuando no existe ninguna secuencia $\langle x_1, x_2, \ldots, x_s \rangle$, de elementos de V, tal que $x_1 = x_s$ y $x_i R x_{i+1}$, para $1 \le i < s$.

3a Describa un algoritmo para decidir si R es acíclica.

Considérese el grafo dirigido $G(V,R)$. La relación R es acíclica sii G no tiene ciclos. Al representar G con una matriz de conectividad A, el problema se reduce a verificar si, para algún $i \in V$, la pareja $(i,i) \in A^+$. Si esto vale, hay un ciclo en el grafo y la relación no es acíclica.

Variante 3a1:

Algoritmo de Warshall

[10/20]

+

"Chequeo de $\neg(i\ A^+\ i), i \in V$"

[10/20]

Variante 3a2:

Calcular A^+ mediante multiplicaciones sucesivas en el semianillo $(M_n(\mathbf{B}),\vee,\wedge,0,1)$.

$A^+ = (\vee i|\ 1\leq i<n\ :\ A^i)$

Más exactamente, sea $B_k = (\vee i|\ 1\leq i<k\ :\ A^i)$, para $1\leq k\leq n$. Entonces $B_1 = A$.

Además, para $1<k<n$:

$$\begin{aligned} B_{k+1} &= (\vee i|\ 1\leq i<k+1\ :\ A^i) \\ &= A(\vee i|\ 0\leq i<k\ :\ A^i) \\ &= A(I \vee (\vee i|\ 1\leq i<k\ :\ A^i)) \\ &= A(I \vee B_k) \end{aligned}$$

Finalmente: $B_n = A^+$.

[10/20]

+

"Chequeo de $\neg(i\ A^+\ i), i\in V$"

[10/20]

Penalización por ineficiencia (3a2)

[-5/20]

3b Estime las complejidades espacial y temporal de su algoritmo.

Variante 3b1:

```
S(n) = O(n^2)       // Matriz de conectividad en k pasos
```

[5/10]

```
T1(n) = O(n^3)      // Complejidad del Alg. de Warshall
T2(n) = O(n)        // chequeo de que ¬(i A+ i), i∈V

T(n)  = O(n^3)
```

[5/10]

Variante 3b2:

```
S(n) = O(n^2)       // Matriz Bk
```

[5/10]

```
T1(n) = n*O(n^3)    // n cálculos de Matriz Bk
      = O(n^4)
T2(n) = O(n)        // chequeo de que ¬(i A+ i), i∈V

T(n)  = O(n^4)
```

[5/10]