ISIS 211 Diseño de Algoritmos
Semestre 2003-2 - Parcial 2
Noviembre 11, 2003
Prof. Rodrigo Cardoso

---

1 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.

1a **[40 puntos]** Utilice programación dinámica para estimar el tiempo mínimo de construcción de una torre de altura `N`, $1 \leq N$. (definición de lenguaje, recurrencia, diagrama de necesidades, invariante). No es necesario que escriba su algoritmo.

1b **[20 puntos]** Estime las complejidades espacial y temporal de su algoritmo en términos de `N`. Para lo temporal, use la asignación como operación básica.

---

2 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 $1 \leq i < s$.

2a **[20 puntos]** Describa un (buen) algoritmo para decidir si `R` es acíclica.
(Buen algoritmo: se califican mejor los algoritmos más eficientes)

2b **[20 puntos]** Estime las complejidades temporal y espacial de su algoritmo.

ISIS 211 Diseño de Algoritmos
Semestre 2003-2 - Parcial 2 Solución
Noviembre 11, 2003
Prof. Rodrigo Cardoso

---

1 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.

1a **[40 puntos]** Utilice programación dinámica para estimar el tiempo mínimo de construcción de una torre de altura `N`, `1≤N`. (definición de lenguaje, recurrencia, diagrama de necesidades, invariante). No es necesario que escriba su algoritmo.

[40/60]

Lenguaje

`tmin(h)` ≈ "No. mínimo de semanas para construir una torre de `h` metros"

[8/40]

`tmin(N) = ?`

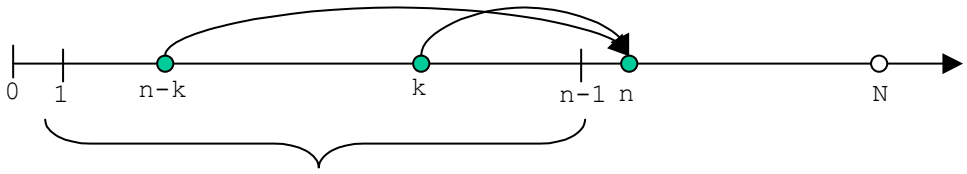
[2/40]

Recurrencia

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

[10/40]

Diagrama de necesidades



[10/40]

Invariante

`Inv: 0<h≤N ∧ (∀i| 0<i<h : TMIN[i] = tmin(i))`

[10/40]

Variante

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

Así, para `h>1`: `tmin(h) = (min k| 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
```

[4/40]

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

[6/40]

Diagrama:

[10/40]

Invariante:

`Inv: 0<h≤N ∧ (∀i| 0<i<h : TMIN[i] = tmin(i))`

[10/40]

1b **[20 puntos]** Estime las complejidades espacial y temporal de su algoritmo en términos de `N`. Para lo temporal, use la asignación como operación básica.

[20/40]

Complejidad espacial:

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

`S(N) = θ(N)`

[5/20]

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:

$$T(N) = (+h|\ 0<h\leq N : \theta(h-1))$$
$$= \theta(+h|\ 0<h\leq N : h-1)$$
$$= \theta(N(N-1)/2)$$
$$= \theta(N^2)$$

[15/20]

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:

$$T(N) = \theta(N(N-1)/4)$$
$$= \theta(N^2)$$

[+5/20]

VARIANTE

Complejidad espacial:

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

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

[5/20]

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:

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

---

2 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 $1\leq i<s$.

2a **[20 puntos]** Describa un (buen) algoritmo para decidir si R es acíclica.
(Buen algoritmo: se califican mejor los algoritmos más eficientes)

[20/20]

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 1:

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

$A^+ = (\vee i \mid 1 \le i < n : A^i)$

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

Además, para $1 < k < n$:

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

Finalmente: $B_n = A^+$.

[15/20]

Variante 2:

Algoritmo de Warshall

[15/20]

+

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

[5/20]

Variante 3:

Algoritmo de Dijkstra en $(M_n(\mathbf{B}), \vee, \wedge, 0, 1)$

[15/20]

con

parada forzada si " i es alcanzable$^+$ desde i"

[5/20]

2b [20 puntos] Estime las complejidades temporal y espacial de su algoritmo.

[20/20]

Variante 1:

Complejidad espacial:

Una matriz $n \times n$ adicional.

$S(n) = \theta(n^2)$

[5/20]

Complejidad temporal:

$$\begin{aligned} T(n) &= (n-1)\theta(n^3) \\ &= \theta(n^4). \end{aligned}$$

[15/20]

Variante 2:

Complejidad espacial:

Una matriz $n \times n$ adicional.

$S(n) = \theta(n^2)$

[5/20]

Complejidad temporal:

$T(n) = \theta(n^3) + \theta(n)$

$= \theta(n^3)$.

[15/20]

Variante 3:

Complejidad espacial:

Una vector $1..n$ adicional.

$S(n) = \theta(n)$

[5/20]

Complejidad temporal:

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

$= \theta(n^3)$.

[15/20]