Solución del Primer Parcial de Programación 3

2 de noviembre de 2015

Nota previa: los logaritmos utilizados a lo largo de esta solución son en base 2.

Ejercicio 1

1. **a**) En el peor caso la condición del `if` en la línea 5 siempre es `true` y la sentencia 6 se ejecuta siempre. En esta condición se compara una entrada con la transpuesta de la matriz.

   Dado que las instrucciones 3 y 4 nos indican que se recorre la parte triangular superior de la matriz, entonces en el peor caso, la matriz debe ser simétrica.

   b) En el mejor caso, la sentencia 6 no se ejecuta nunca. Según lo visto en la parte anterior, esto pasa cuando para ninguna entrada fuera de la diagonal principal de la matriz, una entrada es igual a su simétrica (transpuesta).

   c) Por la parte anterior:

   $$T_B(n) = 0$$

   En el peor caso se tiene que $2 \log n$ se ejecuta en la instrucción 6 en todas las iteraciones. El primer `for` va de 0 a $n-1$ y el segundo de $n-1$ a $c+1$, entrando siempre al `if`:

$$\begin{aligned}
T_W(n) &= \sum_{c=0}^{n-1} \sum_{f=c+1}^{n-1} 2 \log n \\
&= 2 \log n \sum_{c=0}^{n-1} \sum_{f=c+1}^{n-1} 1 \\
&= 2 \log n \sum_{c=0}^{n-1} (n - c - 1) \\
&= 2 \log n (n^2 - \sum_{c=0}^{n-1} c - n) \\
&= 2 \log n (n^2 - \frac{(n-1)n}{2} - n) \\
&= \cancel{2} \log n (\frac{\cancel{2}n^2 - \cancel{n^2} + \cancel{n} - \cancel{2}n}{\cancel{2}}) \\
&= n(n-1) \log n \\
&\Rightarrow T_W(n) \in \Theta(n^2 \log n)
\end{aligned}$$

2. **a**) **Falso**. Supongamos que se cumple $n^2 \in O(n \log n)$. Entonces, por definición, para algún $c \in \mathbb{R}^+$, y $n_0 \in \mathbb{N}$, se cumple que $\forall n \geq n_0, n^2 \leq cn \log n$. Es decir, $\frac{n}{\log n} \leq c$. Dado que $\frac{n}{\log n} \to \infty$, entonces $\forall n_0, \exists n \in \mathbb{N}$ tal que $c < \frac{n}{\log n}$, llegando entonces a un absurdo. Se concluye que $n^2 \notin O(n \log n)$.

   b) **Verdadero.**

   - Sea $c = 1 \Rightarrow 2^n < 3^n c, \forall n \geq 1 = n_0 \Rightarrow 2^n \in O(3^n)$
   - Supongamos que $\exists c > 0$ tal que $\forall n \geq n_0$, $2^n > 3^n c \Rightarrow \forall n \geq n_0$, $(\frac{2}{3})^n > c$. Esto es una contradicción dado que $c > 0$ es constante y $(\frac{2}{3})^n \to 0 \Rightarrow 2^n \notin \Omega(3^n)$

Ejercicio 2

a) 1) El código contiene los siguientes errores:

I. no se inicializa el vector de visitados en `false` (en *C* no se inicializan automáticamente las variables locales a funciones),
II. se registra el orden topológico en la etapa de comienzo de procesamiento de los vértices (preorden), y
III. el orden topológico registrado se encuentra el orden inverso.

2) Los errores se pueden corregir según la numeración anterior:

I. se puede corregir agregando entre las líneas 6 y 7 el código:

```
for (int v = 0; v < N; v++)
  visitado[v] = false;
```

Alternativamente es válida la corrección (de *C++*) que implica modificar la línea 4 a:

```
bool visitado[N] = {false};
```

II. se puede corregir trasladando las líneas 16 y 17 luego de la línea 24; lo que haría que se registre el orden en la etapa de fin de procesamiento (postorden).
III. se puede corregir modificando las líneas 6 y 17 respectivamente a:

```
int pos = N-1;
pos--;
```

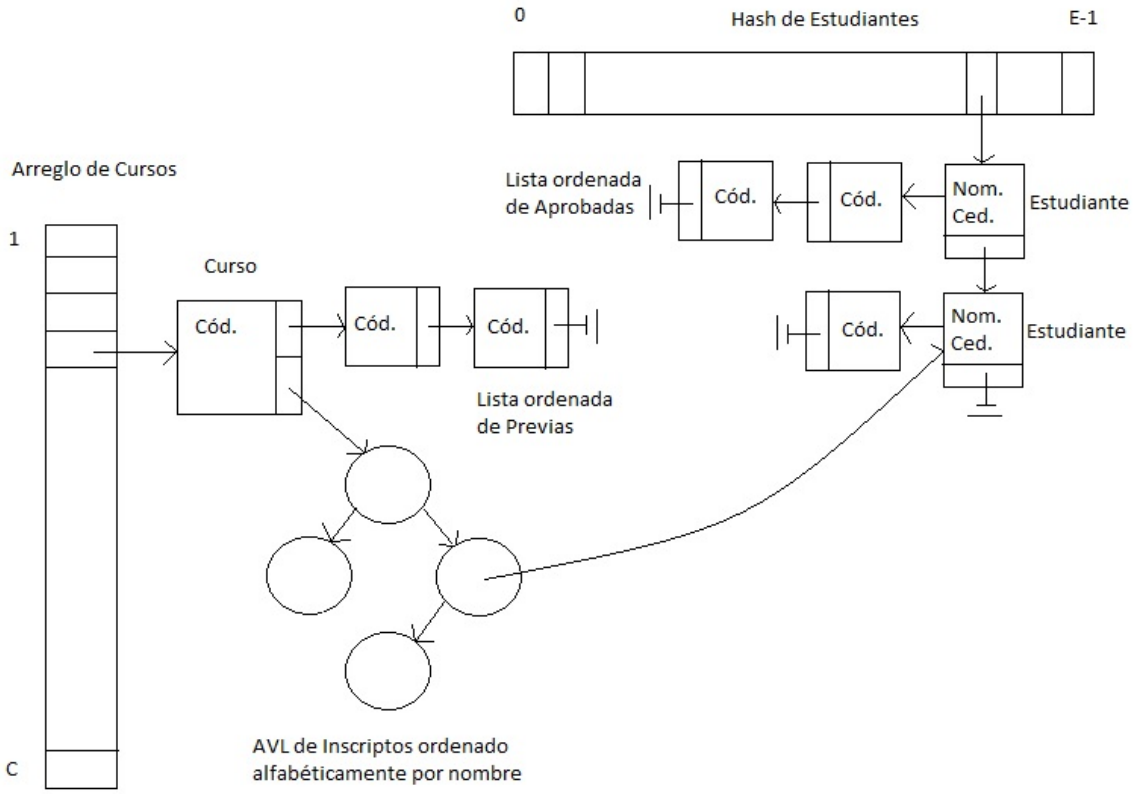
b) 1) Se puede obtener como la diferencia de aristas entre el grafo completo y el anillo. La cantidad de aristas del grafo completo se puede obtener contando las aristas según la secuencia de vértices. El primer vértice tiene $n-1$ aristas, el segundo vértice tiene $n-2$ aristas que no se han contado, y así sucesivamente hasta el último vértice que tiene una arista sin contar. La serie $(n-1)+(n-2)+\cdots+1$ suma $\frac{n(n-1)}{2}$ (alternativamente como las combinaciones de n tomadas de a dos; $\binom{n}{2}$). Por otra parte el anillo contiene n aristas. Por lo que el grafo contiene $\frac{n(n-1)}{2} - n = \frac{n(n-3)}{2}$ aristas.

2) El árbol de cubrimiento contiene n vértices por lo tanto contiene $n-1$ aristas. Un recorrido BFS parte de un vértice inicial, el que se encuentra en el primer nivel, luego alcanza a todos los vértices adyacentes, en un segundo nivel, excepto a los dos vértices que no son adyacentes al inicial. Luego los vértices adyacentes son adyacentes a todos los vértices al que el vértice inicial no es adyacente, por lo que en un tercer nivel de recorrida se visitan todos los vértices. Por lo tanto toda recorrida BFS tiene tres niveles.

3) Sólo hay aristas de niveles adyacentes entre el segundo nivel y el tercer nivel. El tercer nivel tiene dos vértices (los que no son adyacentes al vértice inicial). Los vértices del tercer nivel tienen una arista árbol incidente y una arista de igual nivel que los conecta. Todos los vértices del grafo tienen $n-3$ aristas incidentes. Por lo que cada vértice de tercer nivel tiene una cantidad de aristas de niveles adyacentes igual a la cantidad de aristas incidentes menos la arista del árbol y la arista de igual nivel, es decir $n-3-1-1=n-5$. Dado que son dos vértices, la cantidad de aristas de niveles adyacentes es $2(n-5)$.

Ejercicio 3

Diagrama



Descripción de la multi-estructura

- Hash de estudiantes. Tamaño **E**. La clave es la cédula de cada estudiante.
- Arreglo de cursos. Tamaño **C**. Cada código se identifica con un único lugar dentro del arreglo.
- Curso: tiene como atributo el código, una lista ordenada de códigos de las previas y un AVL con los estudiantes inscriptos, ordenados alfabéticamente según su nombre.
- Estudiante: tiene como atributos su nombre y su cédula y una lista ordenada de códigos de cursos aprobados.
- Cada nodo del AVL tiene un puntero a la estructura del estudiante correspondiente.

Tiempos requeridos para las operaciones

Operación 1 ($O(A)$ caso promedio):

a. Búsqueda de estudiante en el Hash: $O(1)$ promedio.
b. Inserción en lista ordenada del curso aprobado: $O(A)$ promedio.

Por lo tanto la operación en su totalidad cumple ser $O(A)$ en caso promedio.

Operación 2 ($O(E + \log N)$ peor caso):

a. Búsqueda de estudiante en el Hash: $O(E)$ peor caso.
b. Búsqueda de curso en el arreglo: $O(1)$ peor caso.
c. Inserción en AVL del Estudiante según su nombre: $O(\log N)$ peor caso.
d. Link desde nodo AVL a Estudiante: $O(1)$ peor caso.

Por lo tanto la operación en su totalidad cumple ser $O(E + \log N)$ peor caso.

Operación 3 ($O(N \times \text{máx}\{A, P\})$ peor caso):

a. Búsqueda de curso en el arreglo: $O(1)$ peor caso.
b. Recorrer todo AVL en orden: $O(N)$ peor caso.
c. Para cada inscripto recorro la lista de aprobados y la lista de previas del curso a la vez. Con esto se verifica que estén todos los códigos de las previas dentro de la lista de cursos aprobados por el estudiante. Ambas están ordenadas, por lo que si se recorre la lista de aprobados y se llega a un curso con código mayor a la previa buscada, significa que la misma no está aprobada por el estudiante. Notar que cada lista se recorre una única vez. Por lo tanto el orden es A o P en el peor caso, A cuando se terminan de recorren las aprobadas antes que las previas y P en el caso contrario. Nos quedamos con la cota superior, es decir: $\text{máx}\{A, P\}$.

Por lo tanto la operación en su totalidad cumple ser $O(N \times \text{máx}\{A, P\})$ peor caso. La multiplicación se debe a que la búsqueda en las listas ordenadas se realizada para cada estudiante.

Operación 4 ($O(P)$ peor caso):

a. Búsqueda de curso en el arreglo: $O(1)$ peor caso.
b. Inserción en lista ordenada de la previa: $O(P)$ peor caso.

Por lo tanto la operación en su totalidad cumple ser de $O(P)$ peor caso.