Solución Examen de Programación 3 y III

12 de diciembre de 2014

Notas previas:
Los logaritmos utilizados a lo largo de esta solución son en base 2.
$\mathbb{R}^* = \mathbb{R}^+ \cup \{0\}$.

Ejercicio 1 (10 puntos)

a) **VERDADERO**
Por definición de Θ: $\Theta(1) = O(1) \cap \Omega(1)$
Por definición de O: $O(1) = \{g : \mathbb{N} \to \mathbb{R}^* \ / \ \exists c_1 \in \mathbb{R}^+,\ \exists n_1 \in \mathbb{N},\ \forall n \in \mathbb{N},\ n > n_1, \quad g(n) \leq c_1\}$
Por definición de Ω: $\Omega(1) = \{g : \mathbb{N} \to \mathbb{R}^* \ / \ \exists c_2 \in \mathbb{R}^+,\ \exists n_2 \in \mathbb{N},\ \forall n \in \mathbb{N},\ n > n_2, \quad g(n) \geq c_2\}$
Se tiene: $\operatorname{sen}(n) + 2 : \mathbb{N} \to [1, 3]$
Tomando $c_1 = 3$, $c_2 = 1$, $n_1 = n_2 = 0$ se cumple que:

- $\forall n > n_1,\ \operatorname{sen}(n) + 2 \leq c_1$, entonces $\operatorname{sen}(n) + 2 \in O(1)$
- $\forall n > n_2,\ \operatorname{sen}(n) + 2 \geq c_2$, entonces $\operatorname{sen}(n) + 2 \in \Omega(1)$

Entonces $\operatorname{sen}(n) + 2 \in \Theta(1)$.

b) **VERDADERO**
Por definición de O:

- $O(n \log n) = \{g : \mathbb{N} \to \mathbb{R}^* \ / \ \exists c_1 \in \mathbb{R}^+,\ \exists n_1 \in \mathbb{N},\ \forall n \in \mathbb{N},\ n > n_1, \quad g(n) \leq c_1 \cdot n \log n\}$
- $O(n^2) = \{g : \mathbb{N} \to \mathbb{R}^* \ / \ \exists c_2 \in \mathbb{R}^+,\ \exists n_2 \in \mathbb{N},\ \forall n \in \mathbb{N},\ n > n_2, \quad g(n) \leq c_2 \cdot n^2\}$

$O(n \log n) \subsetneq O(n^2)$ significa:

I) $\forall g \in O(n \log n) \implies g \in O(n^2)$
II) $\exists g \in O(n^2),\ g \notin O(n \log n)$

I) Sea $g : \mathbb{N} \to \mathbb{R}^* \ / \ g \in O(n \log n)$, y sean $c_1 \in \mathbb{R}^+, n_1 \in \mathbb{N} \ / \ \forall n > n_1, g(n) \leq c_1 \cdot n \log n$.
Tomando $c_2 = c_1$ y $n_2 = n_1$, tenemos que:

$$\begin{aligned} & \forall n > n_1,\ c_1 \cdot n \log n \leq c_2 \cdot n^2 \\ \underset{c_1 = c_2}{\iff} & \forall n > n_1,\ n \log n \leq n^2 \\ \underset{cancel.}{\iff} & \forall n > n_1,\ \log n \leq n \end{aligned}$$

y esto se cumple ya que $\forall n \in \mathbb{N}^*,\ \log n < n$
Entonces, por transitiva:

$$\forall n > n_2,\ g(n) \leq c_2 \cdot n^2 \implies g \in O(n^2)$$

II) Sea $g = n^2$. $g \in O(n^2)$ (se prueba trivialmente tomando $c_2 = 1$ y $n_2 = 0$).
Supongamos por absurdo que $n^2 \in O(n \log n)$.

$$\begin{aligned} & \implies \exists c_1 \in \mathbb{R}^+, n_1 \in \mathbb{N} \ / \ \forall n > n_1,\ n^2 \leq c_1 \cdot n \log n \\ & \implies \lim_{n \to +\infty} n^2 \leq \lim_{n \to +\infty} c_1 \cdot n \log n \\ & \implies \lim_{n \to +\infty} \frac{n^2}{c_1 \cdot n \log n} \leq \lim_{n \to +\infty} 1 \\ & \implies +\infty \leq 1 \end{aligned}$$

lo que es absurdo.

Ejercicio 2 (10 puntos)

a)
```
void InsertionSort (int* A, int n) {
    int i, j, first;
    for (i = 1; i < n; i++) {
        first = A[i];
        j = i-1;
        while (j >= 0 && first < A[j]) {
           A[j+1] = A[j];
           j--;
        }
        A[j+1] = first;
    }
}
```

b) El peor caso se da en un arreglo ordenado de forma decreciente sin repetidos.

c) En este caso, se tiene que colocar al inicio del arreglo cada elemento a ordenar, en cada paso se realizan la máxima cantidad de comparaciones (i comparaciones en el paso i, $\forall i \in \{1, \ldots, n-1\}$).

$$T_W(n) = \sum_{i=1}^{n-1} i = \frac{n(n+1)}{2} - n = \frac{n^2 - n}{2} = \frac{n(n-1)}{2}$$

Ejercicio 3 (10 puntos)

$C =$

	1	**2**	**3**	**4**	**5**
1	∞	5	3	∞	1
2	5	∞	2	3	∞
3	3	2	∞	1	∞
4	∞	3	1	∞	5
5	1	∞	∞	5	∞

Matriz de costos

Paso	Formalización
1	$S = \{2\}$ $V - S = \{1, 3, 4, 5\}$ $D = [5, \infty, 2, 3, \infty]$
2	$min\{D[1], D[3], D[4], D[5]\} = min\{5, 2, 3, \infty\} = 2$ $\Rightarrow$ Se agrega el vértice 3. $min\{5, 2+3\} = 5$, $min\{3, 2+1\} = 3$, $min\{\infty, 2+\infty\} = \infty$ $S = \{2, 3\}$ $V - S = \{1, 4, 5\}$ $D = [5, \infty, 2, 3, \infty]$

3	$min\{D[1], D[4], D[5]\} = min\{5, 3, \infty\} = 3$ $\Rightarrow$ Se agrega el vértice 4. $min\{5, 3 + \infty\} = 5$, $min\{\infty, 3 + 5\} = 8$ $S = \{2, 3, 4\}$ $V - S = \{1, 5\}$ $D = [5, \infty, 2, 3, \mathbf{8}]$
4	$min\{D[1], D[5]\} = min\{5, 8\} = 5$ $\Rightarrow$ Se agrega el vértice 1. $min\{8, 5 + 1\} = 6$ $S = \{1, 2, 3, 4\}$ $V - S = \{5\}$ $D = [5, \infty, 2, 3, \mathbf{6}]$
5	$min\{D[5]\} = min\{6\} = 6$ $\Rightarrow$ Se agrega el vértice 5. $S = \{1, 2, 3, 4, 5\}$ $V - S = \emptyset$ $D = [5, \infty, 2, 3, 6]$

Ejercicio 4 (10 puntos)

El grafo es conexo y no dirigido. Esto implica que la recorrida comenzada en cualquier vértice alcanzará todos los vértices del grafo. Por lo tanto no es necesaria una función invocadora.

```
BFS (G: Grafo; v: vértice)
    Q     : cola de vértices
    u, w  : vértice  // [1..n]
    T     : Grafo    // array bidimensional n X n
Comienzo
    Para todo w en [1..n] inicializar w como No Marcado
    CrearGrafo(T)
    CrearCola(Q)
    Marcar v
    Encolar(Q,v)
    Mientras No-Vacia(Q)
        u = Primero(Q)
        Q = Desencolar(Q)
        Para todo w en [1..n] tal que (G[u][w] == 1) y (w No Marcado)
            Marcar w
            Encolar(Q,w)
            T[u][w] = T[w][u] = 1 //porque es no dirigido
        Fin Para
    Fin Mientras
    Retornar T
Fin
```

Problema 1 (30 puntos)

a)
- **Forma de la solución**
  Tupla de largo fijo $n = |V|$, $t =< x_0, \cdots, x_{n-1} >$, donde x_i es el color asignado al nodo i, con $0 \leq i \leq n-1$.
- **Restricciones explícitas**
  El color del vértice i debe ser uno de los disponibles: $x_i \in C$, con $0 \leq i \leq n-1$.
- **Restricciones implícitas**
  Los nodos adyacentes tienen distinto color: $\forall i, j \in \{0, \ldots, n-1\},\ (i,j) \in E \implies x_i \neq x_j$.
- **Función objetivo**
  El objetivo es maximizar la agradabilidad, minimizando la cantidad de colores:

  $$f = max_{t \in MinCol}(g(t)), \text{ con } g(t) = \sum_{i=0}^{n-1} D[i] \times A[x_i]$$

  siendo $MinCol = \{t =< x_0, \cdots, x_{n-1} > | \, t \in T \wedge \forall t_1 \in T,\ cantColores(t) \leq cantColores(t_1)\}$

  (t es solución y emplea cantidad mínima de colores)

  $$T = \{t =< x_0, \cdots, x_{n-1} > | \, t \text{ es solución}\}$$

  $$\text{donde } cantColores(t) = \sum_{i=0}^{k-1} pertenece(t, c_i) \text{ y } pertenece(t, c_i) = \begin{cases} 1 \text{ si } c_i \in t \\ 0 \text{ sino} \end{cases}$$
- **Predicados de poda**

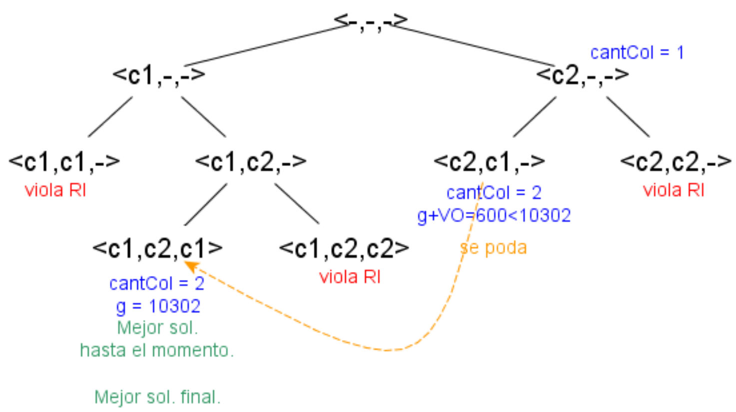
- Si la cantidad de colores utilizados es igual a la de la mejor solución, y la ganancia acumulada hasta la componente i, más el valor más optimista para las demás componentes no supera a la de la mejor solución hasta el momento, se descarta, siendo el valor más optimista $\sum_{j=i+1}^{n-1} D[j] \times max_{l\in\{0,\ldots,k-1\}}(A[c_l])$.
  Formalmente, sea $t =< x_0, \ldots, x_i, -, \ldots, - >$ el prefijo de tupla construido, y t_M la mejor solución hasta el momento. t se descarta si $cantColores(t) = cantColores(t_M)$ y

$$\sum_{j=0}^{i} D[j] \times A[x_j] + \sum_{j=i+1}^{n-1} D[j] \times max_{l\in\{0,\ldots,k-1\}}(A[c_l]) \leq g(t_M)$$

b)
```
// Los siguientes valores se suponen globales:
// int k      : cantidad de colores disponibles
// float* D   : funcion D:V->R (array de largo n)
// float* A   : funcion A:C->R (array de largo k)
// float maxA : maximo A[i] (i=0..k-1)
// -1 indica que no hay color asignado
// Se cuenta con una función que indica si
// dos nodos i y j son adyacentes en el grafo no dirigido G:
// bool adyacentes(int i, int j)
//
// optimistaRestante de una tupla t se inicializa así:
// t.optimistaRestante = 0;
// for (int i = 0; i < t.n; i--) {
//     t.optimistaRestante += maxA*D[i];
// }

struct tupla {
    int* colores;
    int* cantVecesColores;
    int cantColores;
    int agradabilidad;
    int n; // cantidad de nodos de G
    int optimistaRestante;
}

bool PrefijoValido(tupla t, int i) {
    bool ok = true;
    int j = 0;
    // verifico solamente el último elemento del prefijo
    while (ok && j < i-1)) {
        if adyacentes(i-1, j) {
            ok = t.colores[i-1] != t.colores[j];
        }
        j++;
    }
    return ok;
}

bool Poda(tupla t, int i, tupla sol) {
    return (t.cantColores > sol.cantColores
              || (t.cantColores == sol.cantColores
                  && t.agradabilidad + t.optimistaRestante <= sol.agradabilidad))
}

bool EsSolucion(tupla t, int i) {
    return (i == t.n);
}
```

```
void AsignarColor(tupla &t, int i, int color) {
    t.colores[i] = color;
    t.cantVecesColores[color]++;
    if (t.cantVecesColores[color] == 1) {
        t.cantColores++;
    }
    t.agradabilidad += D[i]*A[color];
    t.optimistaRestante -= D[i]*maxA;
}

void DesasignarColor(tupla &t, int i, int color) {
    t.colores[i] = -1;
    t.cantVecesColores[color]--;
    if (t.cantVecesColores[color] == 0) {
        t.cantColores--;
    }
    t.agradabilidad -= D[i]*A[color];
    t.optimistaRestante += D[i]*maxA;
}
```

c) A continuación se muestra el espacio de soluciones.



La mejor solución al problema es $< c_1, c_2, c_1 >$.

Problema 2 (30 puntos)

a) Luego de aplicar una recorrida DFS en G, y considerando el árbol de cubrimiento $T = (V, A_T)$ generado, las aristas de G se pueden clasificar en:

- *aristas tree*: aristas $\in A_T$
- *aristas back*: aristas $\in A - A_T$, que van desde un nodo a un ancestro suyo en T
- *aristas cross*: aristas $\in A - A_T$, que van entre nodos de distintos subárboles de T
- *aristas forward*: aristas $\in A - A_T$, que van desde un nodo a un descendiente suyo en T (como las aristas mencionadas no pertenecen al árbol, no apuntarán a hijos del nodo)

b)
```
void DFS(Vertice v, Bool[] visitados){
    visitados[v] = true;
    //Preprocesamiento
    Para cada w adyacente a v
    Si no visitados[v]
        DFS(w)
    Fin Para
    //Posprocesamiento
}
```

c) Obs.: Cada arista (u,v) se clasifica según los valores prenum y posnum de sus nodos:

- *back* si $prenum(u) > prenum(v)$ y $posnum(v) == -1$
- *forward* si $prenum(u) < prenum(v)$ y $posnum(v) \mathrel{!}= -1$
- *cross* si $prenum(u) > prenum(v)$ y $posnum(v) \mathrel{!}= -1$
- *tree* si $prenum(v) == -1$

Notar que en DFS las aristas cross solo pueden ir "hacia la izquierda".

Nota: en esta implementación el nodo i se va a corresponder con el int $i - 1$.

```
void DFS(Vertice v, bool[] visitados, int[] prenum, int[] posnum, ListaArista &tree,
         ListaArista &back, ListaArista &forward, ListaArista &cross) {
    visitados[v] = true;
    prenum[v] = c1;
    c1++;
    for (int w = 0; w < n; w++) {
        if (adyacentes(v, w)) {
            if (!visitados[w]) {
                //prenum[w] == -1
                agregarArista(tree, v, w);
                DFS(w, visitados, prenum, posnum, tree, back, forward, cross);
            } else { // w está visitado
                if (prenum[v] > prenum[w]) {
                    if (posnum[w] == -1) {
                        agregarArista(back, v, w);
                    } else {
                        agregarArista(cross, v, w);
                    }
                } else {
                    agregarArista(forward, v, w);
                }
            }
        }
    }
    posnum[v] = c2;
    c2++;
}
```