Examen de Programación 3 y III (12/12/2013)

Instituto de Computación, Facultad de Ingeniería, UdelaR

i. Este examen dura 4 horas y contiene **2** carillas. El total de puntos es 100 y se requieren 60 para su aprobación.
ii. En los enunciados llamamos C* a la extensión de C al que se agrega el operador de pasaje por referencia &, y las sentencias *new*, *delete*, el uso de *cout* y *cin y el tipo bool*.
iii. NO se puede utilizar ningún tipo de material de consulta.
iv. No se contestarán dudas durante la última media hora.

Se requiere:

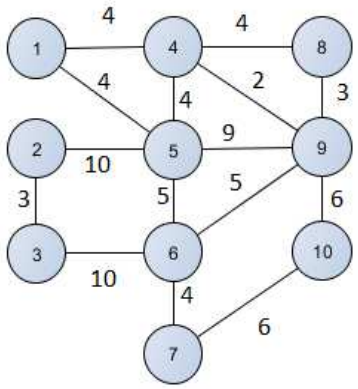
- Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
- Utilizar las hojas de un solo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Poner en la primera hoja la cantidad de hojas entregadas, y entregar el **índice** indicando en qué hoja se respondió cada problema.

Parte Obligatoria

Esta parte es eliminatoria, para la aprobación del examen debe obtenerse un mínimo del **50% de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas.

Ejercicio 1 (10 puntos)

a) Enuncie y demuestre la propiedad MST (Minimum Spanning Tree).
b) Dado el siguiente grafo:



pruebe que existe un árbol de cubrimiento de costo mínimo que contiene la arista (*2*,*3*), utilizando la parte a).
Nota: esta parte no se corregirá si no se hizo la parte a) de forma satisfactoria.

Solución

a)

Propiedad MST (**M**inimum **S**panning **T**ree):

Sea $G = (V, E)$ un grafo conexo con una función de costo definido sobre las aristas.

Sea U un subconjunto de los vértices V, si (u, v) es una arista de costo mínimo tal que $u \in U$ y $v \in V - U$ entonces existe un árbol de cubrimiento de costo mínimo que incluye a (u, v) como una de sus aristas.

Demostración:

Para demostrar la propiedad se supondrá por absurdo que no existe ningún árbol de cubrimiento de costo mínimo para el grafo G que incluya la arista (u, v) entre sus aristas.

Sea entonces, T cualquier árbol de cubrimiento de costo mínimo para G. Si se agrega la arista (u, v) a T, se genera un ciclo (recordar que T es un árbol y por lo tanto existe un camino de u a v).

Debe existir otra arista (u', v') tal que $u' \in U$ y $v' \in V - U$, y que forme parte del camino de u a v.
Si se quita la arista (u', v') se rompe el ciclo y se tiene un árbol de cubrimiento T' en el cual se cumple:

$$Costo(T') = Costo(T) - c(u', v') + c(u, v)$$

Y como (*u*, *v*) es de costo mínimo (entre las aristas que tienen un extremo en *U* y el otro en *V-U*) $\Rightarrow c(u,v) \leq c(u',v')$ $\Rightarrow Costo(T') \leq Costo(T)$
lo cual contradice la hipótesis de que no existe ningún árbol de cubrimiento de costo mínimo que incluya la arista (*u*, *v*).

b)
Si consideramos *U = {1,2,4,5,6,7,8,9,10}* , hay dos aristas que unen a los conjuntos *U* y *V-U* . Una de costo mínimo es la arista *e = {2,3}* . Por lo tanto por la propiedad MST hay un árbol de recubrimiento de costo mínimo que la incluye.

Ejercicio 2 (10 puntos)

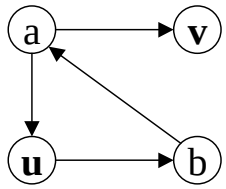
Demuestre o de un contraejemplo para el siguiente enunciado: Para todo grafo *G=(V,E)* dirigido, si existe un camino dirigido entre dos vértices *u* y *v* de *G* y si *u* es visitado antes que *v* en una recorrida *DFS*, entonces *v* es descendiente de *u* en alguno de los árboles del bosque de la recorrida *DFS*.

Solución

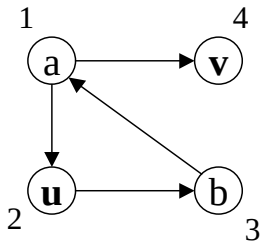
Considerando que el enunciado puede interpretarse como que es verdadero, dado que puede entenderse que **existe** por lo menos una recorrida *DFS* sobre el grafo en la cual si *u* es visitado antes que *v*, entonces *v* es descendiente de *u* en alguno de los árboles del bosque de la recorrida *DFS*, se acepta como válida tal respuesta si se demuestra correctamente, indicando de alguna manera genérica cual o cómo es la recorrida *DFS* que hace verdadero al enunciado.

Si se considera que el enunciado se refiere a **todas** las recorridas *DFS* donde *u* es visitado antes que *v*, el enunciado es falso, tal como se muestra en el siguiente contraejemplo:

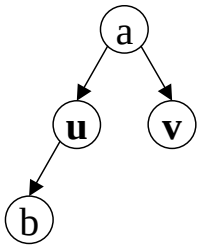
Si consideramos el siguiente grafo:



Existe un camino de *u* a *v* el cual es: *u, b, a, v*. Un DFS partiendo de *a* podría ser:



En el cual el vértice *u* es visitado antes que *v*, pero su árbol asociado es:



Donde el vértice *v* no es descendiente del vértice *u*.

Ejercicio 3 (10 puntos)

Implemente en el lenguaje C* el algoritmo ***Selection Sort***. Calcule el costo en el peor, mejor y caso promedio. En términos de complejidad, ¿el algoritmo es óptimo si se hace un análisis basado en comparaciones?

Solución

```
void selectionSort (int* a, int n){
	int min, tmp;
	for (int i = 0; i < n-1; i++){
		min = i;
		for (int j = i + 1; j < n; j++){
			if (a[j] < a[min])
				min = j;
		}
		// Intercambio de elementos
		tmp = a[i];
		a[i] = a[min];
		a[min] = tmp;
	}
}
```

El algoritmo para una entrada de tamaño *n* siempre realiza la misma cantidad de comparaciones.

$$T(n)=\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}1=\sum_{i=0}^{n-2}(n-1-(i+1)+1)=\sum_{i=0}^{n-2}(n-i-1)=\sum_{i=0}^{n-2}(n-1)-\sum_{i=0}^{n-2}i=(n-1)(n-1)-\frac{(n-2)(n-1)}{2}$$

Resumiendo:

$T(n)=\frac{(n-1)n}{2}$.

El algoritmo no es óptimo ya que realiza $\Theta(n^2)$ comparaciones y por resultado del curso los algoritmos de complejidad óptima basada en comparaciones son de orden $\Theta(n\log n)$.

Ejercicio 4 (10 puntos)

Dadas las funciones arbitrarias $f:N\to R^*, g:N\to R^*$ y $h:N\to R^*$, partiendo de las definiciones de límite, θ,Ω y O:

a) Pruebe que si el $\lim_{n\to+\infty}\frac{f(n)}{g(n)}=0$ entonces $f(n)\in O(g(n))$ y $g(n)\notin O(f(n))$

b) Pruebe que si el $\lim_{n\to+\infty}\frac{f(n)}{g(n)}=+\infty$ entonces $f\notin O(g)$ y $g\in O(f)$. Puede suponer que para todo *n*, $f(n)\neq 0$ y $g(n)\neq 0$. Para esta parte puede utilizar propiedades de límite además de las definiciones.

Solución

*1) Desarrollo de definición de $f(n)\in O(g(n))$: $(\exists K\in R^+)(\exists n_0\in N)(\forall n\in N)\left(n\geq n_0\to f(n)\leq K*g(n)\right)$

*2) Desarrollo de definición de $\lim_{n\to+\infty}\frac{f(n)}{g(n)}=0$

Sii (definición límite de sucesiones)

$$(\forall\varepsilon\in R^+)(\exists n_1\in N)(\forall n\in N)\left(n\geq n_1\to\left|\frac{f(n)}{g(n)}-0\right|<\varepsilon\right)$$

Partiendo de *2), operando y teniendo en cuenta que *f* y *g* son funciones positivas se llega a que

$$(\forall\varepsilon\in R^+)(\exists n_1\in N)(\forall n\in N)\left(n\geq n_1\to f(n)<\varepsilon*g(n)\right)$$

Luego, se elige un ε cualquiera. Esto verifica la definición $f(n)\in O(g(n))$ escrito en *1).

Resta probar que $g(n) \notin O(f(n))$

Lo anterior equivale a decir que no se cumple la afirmación
$(\exists K \in R^+)(\exists n_0 \in N)(\forall n \in N)\big(n \geq n_0 \rightarrow g(n) \leq K * f(n)\big)$

Sii (equivalencia lógica)

No existe $K \in R^+$ tal que $(\exists n_0 \in N)(\forall n \in N)\big(n \geq n_0 \rightarrow g(n) \leq K * f(n)\big)$

Para probarlo se supone por absurdo que existe $K \in R^+$ tal que $(\exists n_0 \in N)(\forall n \in N)\big(n \geq n_0 \rightarrow g(n) \leq K * f(n)\big)$.

Por *2)
$(\forall \varepsilon \in R^+)(\exists n_1 \in N)(\forall n \in N)\left(n \geq n_1 \rightarrow \frac{1}{\varepsilon} f(n) < g(n)\right)$

Si tomo el $n_3 = \max(n_0, n_1)$ y $\varepsilon = \frac{1}{K}$ entonces se cumple
$(\forall n \in N)\big(n \geq n_3 \rightarrow K * f(n) < g(n)\big)$ y $(\forall n \in N)\big(n \geq n_3 \rightarrow g(n) \leq K * f(n)\big)$

Lo cual es una contradicción.

2)

Por propiedad de límite si $\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$ entonces $\lim_{n \to +\infty} \frac{g(n)}{f(n)} = 0$. Finalmente, por lo probado en la parte anterior $g \in O(f)$ y $f \notin O(g)$.

Problemas

Problema 1 (30 puntos)

Sea una matriz de enteros de dimensión ***n×m***. Se desea encontrar el máximo y el mínimo de la misma.

Se pide:

a) Construir un algoritmo que utilice la técnica de **Divide & Conquer**.
b) Identifique en su algoritmo donde aplica los distintos ítems de la técnica (división, combinación, paso base y paso recursivo).
c) Determine cual es la operación básica de su algoritmo.
d) Encuentre la cantidad de operaciones básicas que realiza su algoritmo en el peor caso.

Suponga **n** potencia de algún número conveniente.

Solución

a)

Se supone n potencia de 2.

```
// Devuelve el mayor de los 4 enteros, en el peor caso realiza 3 comparaciones
int Mayor (int max1, int max2,int max3,int max4);

// Devuelve el menor de los 4 enteros, en el peor caso realiza 3 comparaciones
int Menor (int max1, int max2, int max3,int max4);

void MaxMin (Matriz m int i,int j, int n, int & max, int & min)
{
	int m;
```

```
(1)     if (n==1) {
(2)             max=M[i,j];
(3)             min=M[i,j];
(4)     }
(5)     else {
(6)             m = n/2;
(7)             MaxMin(M,i,j,m,max1,min1);
(8)             MaxMin(M,i+m,j,m,max2,min2);
(9)             MaxMin(M,i,j+m,m,max3,min3);
(10)            MaxMin(M,i+m,j+m,m,max4,min4);
(11)            max = Mayor(max1,max2,max3,max4);
(12)            min = Menor(min1,min2,min3,min4);
        }
}
```

b)

División: en las líneas (7) a (10)

Combinación: en las líneas (11) y 12)

Paso base: en las líneas (1) a (3)

Paso recursivo: en las líneas (7) a (10)

c)

La operación básica es la comparación entre elementos.

d)

$$T(1) = 0$$

$$T(n) = 4T\left(\frac{n}{2}\right) + 6$$

$$T(n) = 4T\left(\frac{n}{2}\right) + 6$$

$$4T\left(\frac{n}{2}\right) = 4^2 T\left(\frac{n}{2^2}\right) + 4 \cdot 6$$

$$4^2 T\left(\frac{n}{2^2}\right) = 4^3 T\left(\frac{n}{2^3}\right) + 4^2 \cdot 6$$

...

$$4^{i-1} T\left(\frac{n}{2^{i-1}}\right) = 4^i T\left(\frac{n}{2^i}\right) + 4^{i-1} \cdot 6$$

sumando ambos miembros

$$T(n) = 4^i T\left(\frac{n}{2^i}\right) + 6\sum_{j=0}^{i-1} 4^j$$

si $n = 2^i$ entonces $i = \log_2 n$,

$$T(n) = 2 \cdot (4^i - 1)$$

en definitiva,

$$T(n) = 2 \cdot (n^2 - 1)$$

Problema 2 (30 puntos)

Un estudiante se propone obtener al menos ***M*** créditos en el próximo período. Hay ***n*** actividades distintas con las que puede obtenerlos. Cada actividad tiene una fecha de inicio, una fecha de fin y otorga una cantidad de créditos. El estudiante quiere obtener los créditos haciendo la menor cantidad posible de actividades. Además no quiere que haya superposición en el tiempo entre ninguna de las actividades que elija (si la fecha de fin de una coincide con la de inicio de otra se considera que se superponen). Las actividades se identifican con enteros del 1 al ***n***.

a) Se plantea un algoritmo que consiste en elegir las actividades en orden decreciente de la cantidad de créditos hasta que la cantidad de créditos alcance o supere a ***M***. Al elegir una actividad no se debe superponer con las elegidas anteriormente.

   i. ¿El algoritmo planteado es ávido (Greedy)? Responda **Sí** o **No**.
   ii. Demuestre que, si hay solución, este algoritmo siempre encuentra una solución óptima, o muestre un ejemplo, con la menor cantidad de actividades posibles, de que no siempre la encuentra.

b) Se plantea resolver el problema por **Backtracking**. Los datos están disponibles en el vector *Actividades*, cuyo índice está entre 1 y ***n***. *Actividades[i].Inicio*, *Actividades[i].Fin* y *Actividades[i].Creditos* indican la fecha de inicio, la fecha de fin y los créditos de la actividad *i* respectivamente. Las actividades están ordenadas en el vector de manera creciente según la fecha de inicio (o sea, *Actividades[i].Inicio* $\leq$ *Actividades[i+1].Inicio*).

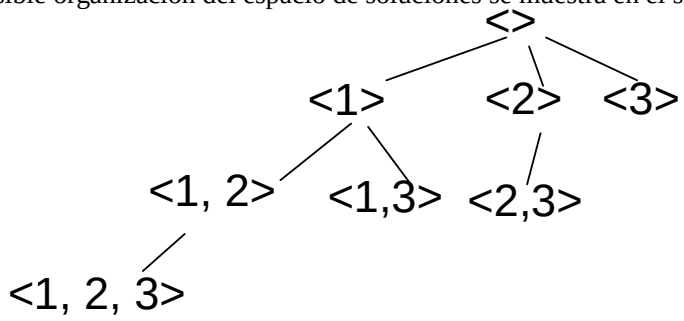
   i. Formalizar el problema en términos de **Backtracking** Describa formalmente y con lenguaje natural, la forma de la tupla solución (l*a forma de la tupla solución debe ser de longitud variable e indicar las actividades elegidas de manera creciente por la fecha de inicio*), las restricciones explícitas e implícitas, la función

objetivo y los predicados de poda.

ii. Considere las siguientes actividades, con una meta de ***M*** = 10 créditos:

Actividad	Inicio	Fin	Créditos
1	1	4	6
2	2	5	10
3	6	9	6

una posible organización del espacio de soluciones se muestra en el siguiente árbol de soluciones:



¿En qué nodos del árbol de soluciones dado se cumplen las restricciones explícitas? ¿En qué nodos del árbol de soluciones dado se cumplen las restricciones implícitas? ¿Qué nodos del árbol de soluciones dado representan soluciones óptimas?

Solución

a)

i. Sí.

ii. El algoritmo en general no encuentra solución óptima. Supongamos que la meta es M = 15 y las actividades son:

	Inicio	Fin	Créditos
1	1	4	8
2	2	6	10
3	5	9	8

El algoritmo propuesto elige la actividad 2 que da 10 créditos y debe descartar las actividades 1 y 3. Por lo tanto no se alcanza el mínimo requerido de 15.
Pero se puede obtener la solución eligiendo las actividades 1 y 3.

b)

i.

Forma de la solución

Tupla t = <t_1, ..., t_h> de largo variable h ≤ ***n***, denotado |t|, cuyos componentes son los identificadores de las actividades elegidas.

Restricciones explícitas

Los componentes de la tupla son los identificadores de actividad:

$t_i \in \{1..n\}, con\ 1 \le i \le h.$

Restricciones implícitas

1. Las actividades se incluyen en orden creciente:
   $t_{i-1} < t_i, con 2 \le i \le h.$

2. Las actividades no se superponen:

   $Actividades[t_{i-1}].Fin < Actividades[t_i].inicio, con 2 \le i \le h.$

Como las actividades están ordenadas en forma creciente por la fecha de inicio alcanza que no haya superposición con la anterior inmediata ya que

$Actividades[t_{i-1}].Fin > Actividades[t_{i-1}].Inicio > Actividades[t_{i-2}].Fin$ si i > 2.

NOTA: Esta restricción contiene a la restricción 1 porque implica

$Actividades[t_{i-1}].inicio < Actividades[t_i].inicio,$

lo que, dado el orden en que están dadas las actividades implica $t_{i-1} < t_i$.

3. La suma de los créditos es al menos *M*:

$$\sum_{i=1}^{h} Actividades[i].Creditos \geq M$$

Función objetivo Minimizar la cantidad de actividades elegidas:
F = min {|t| | t en T}, donde T = {t = <t_1, ..., t_h> / t es solución}.

Predicados de poda

Si los créditos acumulados en el prefijo de tupla <t_1, ..., t_k> más la suma de los créditos de las actividades aún no consideradas es menor a M, entonces se detiene la construcción de la tupla y se la descarta.

$$\sum_{i=1}^{k} Actividades[t_i].Creditos + \sum_{i=t_k+1}^{n} Actividades[i].Creditos < M \ .$$

ii. En todos los nodos del árbol de soluciones se cumplen las restricciones explícitas

Las restricciones implícitas se cumplen en los nodos <2>, <1,3> y <2,3>.
No se cumplen en los nodos <1,2> y <1,2,3> porque hay superposición entre las actividades 1 y 2.
no se cumplen en los nodos <>, <1> y <3> porque la suma de créditos es menor que M.

El único nodo que representa una solución óptima es <2> porque tiene una actividad. Los otros dos nodos <1,3> y <2,3> que representan soluciones tienen dos actividades.

Recordar que: M=10 y

Actividad	Inicio	Fin	Créditos
1	1	4	6
2	2	5	10
3	6	9	6