Solución del Examen de Programación 3 y III (16/07/2015)

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **7** carillas. El total de puntos es 100 y se requieren 60 para su aprobación.
2. 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*.
3. NO se puede utilizar ningún tipo de material de consulta.
4. No se contestarán dudas durante la última media hora.

Se requiere:

- Numerar todas las hojas e incluir en cada una el nombre, la cédula de identidad, el número de hoja y el TOTAL de hojas.
- Utilizar las hojas de un solo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- En la **carátula**: completar el índice indicando en qué hoja se comenzó la respuesta a cada problema/ejercicio y el total de hojas entregadas.

Parte Obligatoria (Ejercicios 1 a 4)

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)

1. Dados dos algoritmos que con una entrada de tamaño **n** tienen un tiempo de ejecución en $\Theta(f(n))$ en todos los casos. Para una entrada de tamaño 100, demorarían tiempos similares? Justifique su respuesta.

Solución:
No necesariamente. Sabemos cómo se comportan en relación con **n** pero no cuánto tardan para un n dado. Por ejemplo, uno de ellos podría tener un tiempo T(n) = n y el otro T(n) = 1000*n, ambos serían $\Theta(n)$ pero para n=100 demorarían tiempos distintos.

2. Demuestre o dé un contraejemplo:
   a) Si $f \in O(g) \Rightarrow f/h \in O(g)$
   b) Si $f \in \Theta(g) \Rightarrow f/h \in \Theta(g/h)$

(notación: $(f/g)(n) = f(n)/g(n)$; $f,g: \mathbb{N} \to \mathbb{R}+$, $g(n) \neq 0$)
Para demostrar debe usar las definiciones. En caso de usar propiedades vistas en el curso deberá demostrarlas.

Solución:
a) Falso. Basta tomar f(n) = g(n) = n y h(n) = 1/n.
b) Verdadero. Demostración:

$f(n) \in O(g(n)) \Leftrightarrow$

$f \in O(g)$
$\Rightarrow$ (def O)
$\exists\, c \in R+, \exists\, n_0 \in N, \forall\, n \in N, n > n_0 , f(n) \leq c*g(n)$
$\Rightarrow$ ($h(n) > 0$)
$f(n)/h(n) \leq c*g(n)/h(n)$
$\Rightarrow$ (def O, def /)
$f/h \in O(g/h)$

Analogamente se prueba que $f/h \in \Omega(g/h)$

Finalmente,
$f/h \in O(g/h)$ y $f/h \in \Omega(g/h)$
$\Rightarrow$ (def Θ)
$f/h \in \Theta(g/h)$

(notación: (f/g)(n) = f(n)/g(n); f,g: $\mathbb{N}$ -> $\mathbb{R}^+$, g(n)≠0)
Para demostrar debe usar las definiciones. En caso de usar propiedades vistas en el curso deberá demostrarlas.

Ejercicio 2 (10 puntos)

1. Defina el concepto de árbol de decisión.
2. Escriba una implementación de **Selection Sort** y diagrame su árbol de decisión para **n=3**.

Solución:

Ver teórico.

Ejercicio 3 (10 puntos)

1. Enunciar y demostrar la propiedad MST (Minimum Spanning Tree).
2. Dar un ejemplo de COMO se aplica dicha propiedad, en el contexto de los algoritmos greedy para hallar el árbol de cubrimiento mínimo de un grafo conexo no dirigido.

Solución:

1. Ver teórico de Greedy.

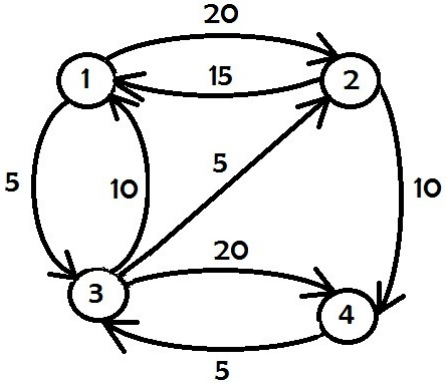
2. La propiedad MST sirve para justificar la CORRECTITUD de los algorimtos Prim y Kruskal de determinación de un árbol de cubrimiento mínimo.

   Tomando como ejemplo el algoritmo de Prim, en este caso la aplicación de la propiedad MST es directa, pues en todo momento se manejan dos conjuntos de vértices, el conjunto U del árbol de cubrimiento en construcción y V-U, y en cada paso se agrega al árbol la arista de menor costo con un extremo en U y otro en V-U, lo cual se justifica por la propiedad MST.

   En el caso de Kruskal, puede realizarse una clasificación “dinámica” de los vertices de V en 2 conjuntos U y V-U a medida que se incorporan aristas al árbol solución, imponiendo que los extremos de cada arista de menor costo elegida que no forme ciclo pertenezcan a conjuntos distintos (puede implicar reclasificar dinámicamente las pertenencias de vértices a los conjuntos). Al obtener el árbol solución, con la clasificación final de los vértices se cumple la hipótesis de la MST para cada paso de Kruskal, lo que justifica el algoritmo.

Ejercicio 4 (10 puntos)

1. Indique porqué es aplicable el principio de optimalidad al problema de hallar los costos de los caminos de mínimo costo entre todo par de vértices de un grafo que tiene costos positivos asociados a sus aristas. Escriba la recurrencia del algoritmo de Floyd para resolver el problema, definiendo completamente todas las estructuras que intervienen en la misma.
2. Explicar la ejecución paso a paso del algoritmo de Floyd para el grafo de la figura.



Solución:

1. Ver teórico de Programación Dinámica.
2. $D^0 = C = \begin{pmatrix} 0 & 20 & 5 & \infty \\ 15 & 0 & \infty & 10 \\ 10 & 5 & 0 & 20 \\ \infty & \infty & 5 & 0 \end{pmatrix}$ matriz de costos del grafo

   $D^1 = \begin{pmatrix} 0 & 20 & 5 & \infty \\ 15 & 0 & 20 & 10 \\ 10 & 5 & 0 & 20 \\ \infty & \infty & 5 & 0 \end{pmatrix}$

   costos de los caminos de menor costo pasando por el vértice 1 como vértice intermedio.

   $D^2 = \begin{pmatrix} 0 & 20 & 5 & 30 \\ 15 & 0 & 20 & 10 \\ 10 & 5 & 0 & 15 \\ \infty & \infty & 5 & 0 \end{pmatrix}$

   costos de los caminos de menor costo pasando por los vértices 1 y 2 como vértices intermedios.

   $D^3 = \begin{pmatrix} 0 & 10 & 5 & 20 \\ 15 & 0 & 20 & 10 \\ 10 & 5 & 0 & 15 \\ 15 & 10 & 5 & 0 \end{pmatrix}$

   costos de los caminos de menor costo pasando por los vértices 1,2 y 3 como vértices intermedios.

   $D^4 = \begin{pmatrix} 0 & 10 & 5 & 20 \\ 15 & 0 & 15 & 10 \\ 10 & 5 & 0 & 15 \\ 15 & 10 & 5 & 0 \end{pmatrix}$

   costos de los caminos de menor costo pasando por los vertices 1,2,3 y 4 como vértices intermedios (solución del problema).

Problemas

Problema 1 (30 puntos)

Un camionero quiere hacer al menos C cantidad de dinero. Hay n pedidos disponibles en la empresa entre los que elegir para lograr su objetivo. A su vez cada pedido tiene una fecha de inicio, una fecha de fin y otorga una cierta cantidad de dinero. El camionero quiere ganar el dinero haciendo la menor cantidad posible de pedidos. Además no quiere que haya superposición en el tiempo entre ninguna de los pedidos que elija (si la fecha de fin de una coincide con la de inicio de otro también se considera que se superponen). Los pedidos se identifican con enteros del 1 al n.

Se plantea resolver el problema por Backtracking. Considere los siguientes datos:

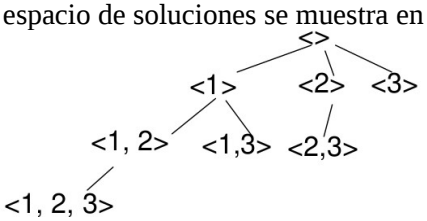
- Los datos están disponibles en el vector Pedidos, cuyo índice está entre 1 y n.
- Pedidos[i].Inicio, Pedidos[i].Fin y Pedidos[i].Dinero indican la fecha de inicio, la fecha de fin y el dinero del pedido i respectivamente.
- Los pedidos están ordenadas en el vector de manera creciente según la fecha de inicio (o sea, Pedidos[i].Inicio ≤ Pedidos[i+1].Inicio).

i. Formalizar el problema en términos de Backtracking. Describa formalmente y con lenguaje natural, la forma de la tupla solución (la forma de la tupla solución debe ser de longitud variable e indicar los pedidos elegidos de manera creciente por la fecha de inicio), las restricciones explícitas e implícitas, la función objetivo y los predicados de poda.

ii. Considere los siguientes pedidos, con una meta de C = 10:

Pedido	Inicio	Fin	Dinero
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:

i. Forma de la solución

Tupla t=<t1,...,th> de largo variable h ≤ n, denotado |t|, cuyos componentes son los identificadores de los pedidos que atendió el camionero.

Restricciones expícitas

Los componentes de la tupla son los identificadores de los pedidos:
ti ∈ {1..n}, con 1 ≤ i ≤ h

Restricciones implícitas

1. Los pedidos se incluyen en orden creciente
   t i − 1 < t i , con 2 ≤ i ≤ h.
2. Los pedidos no se superponen
   Actividade s [t i − 1] . Fin < Actividade s [t i] . inicio, con 2 ≤ i ≤ h.

   Como los pedidos están ordenadas en forma creciente por la fecha de inicio alcanza que no haya superposición con la anterior inmediata ya que
   Actividade s [t i − 1] . Fin > Actividade s [t i − 1] . Inicio > Actividade s [t i − 2] . Fin si i > 2.

   NOTA: Esta restricción contiene a la restrcción 1 porque implica
   Actividade s [t i− 1] . inicio < Actividade s [t i] . inicio ,
   lo que, dado el orden en que están dadas las actividades implica t i − 1 < t i .

Función objetivo

Se debe minimizar la cantidad de pedidos a atender:
F = min {|t| | t en T}, donde T = {t = <t 1 , ..., t h > / t es solución}.

Predicados de poda

Si el dinero acumulado en el prefijo de tupla <t 1 , ..., t k > más la suma del dinoer ode las actividades no consideradas es menor a M, entonces se detiene la construcción de la tupla y se la descarta.
∑ Actividade s [t] . Creditos + ∑ Actividade s [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 los pedidos 1 y 2.
No se cumplen en los nodos <>, <1> y <3> porque la suma de dinero de los pedidos 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

Pedido	Inicio	Fin	Dinero
1	1	4	6
2	2	5	10
3	6	9	6

Problema 2 (30 puntos)

Sea $G = (V,A)$ un grafo con ciclos, no dirigido, conexo, sin lazos ni aristas múltiples. Cada arista $(v_i, v_j) \in A$ tiene asociado un costo positivo $Costo\ (v_i, v_j)$. Se denota con $n = |V|$ la cantidad de vértices de G.

1. Escriba el seudocódigo de $Ciclo\ (G)$ que devuelve un ciclo (como lista de vértices) de G. Es decir, devuelve una lista $(v_1, v_2, \ldots, v_h, v_{h+1})$, con $v_1 = v_{h+1}$ y $(v_i, v_{i+1}) \in A$ para $i \in [1\ .\ .\ h]$.

Si hay más de un ciclo devuelve cualquiera de ellos. Los vértices son de tipo $Vertice$, que es el subrango $[1\ .\ .n]$.

2. Muestre el funcionamiento del algoritmo en el siguiente grafo: $A = \{(1, 2), (2, 3), (3, 4), (3, 5), (5, 6), (6, 2), (6, 7)\}$.
3. Escriba el seudocódigo de MasCostosa (C) que, dada la lista de vértices C que forman un camino no vacío de G, devuelve la arista de mayor costo en C.
4. Asuma que la cantidad de aristas de G es igual a la cantidad de vértices. Escriba el seudocódigo de $ACCM(G)$, que devuelve un árbol de cubrimiento de costo mínimo de G. No puede utilizar un algoritmo que use la propiedad MST vista en el curso.
5. Fundamente en lenguaje natural por qué su algoritmo devuelve un ACCM.

Solución:

1. Se ejecuta una DFS hasta encontrar un vértice ya visitado, lo cual indica que se encontró un ciclo. Como el grafo es conexo, no dirigido y tiene un ciclo se sabe que esto va a ocurrir. Al encontrar el vértice visitado se recorre el camino inverso hasta volver a encontrar ese nodo. Para eso se mantiene el array padres. Este array sirve también para determinar si el vértice fue visitado (el valor -1 significa que no fue visitado).

```
BuscaCiclo (G: Grafo; u: Vertice; padres: ARRAY [1..n] OF INTEGER): Lista
  A ← Adyacentes(G; u)
  encontrado ← FALSE
  WHILE (NOT EsVacia (A)) AND (NOT encontrado) DO
    v ← Primero (A); A ← Resto (A)
    IF v <> padres [u] THEN
      IF padres [v] = -1 THEN
        padres [v] ← u
        L ← BuscaCiclo (G, v, padres)
      ELSE { se encontró un ciclo}
        L ← Cons (v, ListaVacia)
        w ← u
        WHILE w <> v DO
          L ← Cons (w, L)
          w ← padres [w]
```

```
            END WHILE
            L ← Cons (v, L)
            encontrado ← TRUE
          END IF
        END IF
    END WHILE
    IF NOT encontrado THEN
       L ← CrearLista
    END IF
    RETURN L
END BuscaCiclo


Ciclo (G:Grafo)
    padres: ARRAY [1..n] OF INTEGER
    padres [i]  ← -1, para i en  [1 . .  n]
    padres [1] ← 0 {raíz del  árbol de la DFS}
    RETURN BuscaCiclo(G, 1, padres)
END Ciclo
```

2. Se muestra el orden en que se recorren las aristas y lo que sucede con el valor de padre del vértice destino de la arista recorrida.

Arista	Padre
(1,2)	- 1 → 1
(2,3)	- 1 → 2
(3,4)	- 1 → 3
(3,5)	- 1 → 3
(3,6)	- 1 → 5
(6,2)	1

Al llegar por segunda al vértice 2 se encuentra el ciclo. El camino inverso se obtiene con el array padres: 2, 6, 5, 3, 2

3.

```
MasCostosa (C: Lista): Arista
    Precondición: C es un camino no vacío de G
     vi ← Primero(C);  C ← Resto(C)
     vj ← Primero(C);  C ← Resto(C)
     anterior ← vj
    WHILE NOT EsVacia (C) DO
        IF  Costo (anterior, Primero(C)) > Costo(vi, vj) THEN
          (vi, vj) ← (anterior, Primero(C))
        END  IF
        anterior ← Primero(C); C ← Resto(C)
    END WHILE
    RETURN (vi, vj)
end MasCostosa
```

4.

```
ACCM (G: Grafo): Arbol
   Precondición: G tiene n vértices y n aristas
   C ← Ciclo (G)
   (vi; vj) ← MasCostosa (C)
   T ← RemoverArista (G, (vi;,vj))
   RETURN T
end ACCM
```

5. Como el grafo es conexo, sin lazos ni aristas múltiples, que tenga n aristas implica que hay un ciclo. Al remover cualquiera de las aristas de ese ciclo se obtiene un árbol de cubrimiento. De esos árboles el que tiene menor costo es el que se obtiene al remover la arista de mayor costo del ciclo.
El algoritmo propuesto encuentra el ciclo, luego la arista de mayor costo en el ciclo y la remueve del grafo.