Examen de Programación 3 y III (17/12/2012)

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **4** 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. Puede usarse lo dictado en el curso sin demostrarlo, indicando claramente lo que se está usando.
4. 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 un **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. Usted podrá encontrar planteos prácticos. Los mismos deben ser resueltos justificando detalladamente la correspondencia con la base teórica que utilice en su respuesta.

Ejercicio 1 (10 puntos)

a) Enunciar y demostrar la propiedad MST (**M**inimum **S**panning **T**ree).
b) Indique cual es la utilidad de dicha propiedad según lo visto en el curso.

Ejercicio 2 (10 puntos)

a) Considere una recorrida DFS de un grafo dirigido. Defina los siguientes términos con respecto al bosque resultado de la recorrida: arista *tree*, arista *forward*, arista *back* y arista *cross*.
b) Sea $G=(V,E)$ un grafo no dirigido. Considere el grafo dirigido $G'=(V,E')$ con los mismos vértices que G y donde su conjunto de aristas E' contiene las aristas (u,v) y (v,u) por cada arista $(u,v) \in E$. Indique cuales de los cuatro tipos de aristas mencionados en la parte a) pueden ocurrir en una recorrida DFS de G'. Justifique detalladamente.

Ejercicio 3 (10 puntos)

a) En el contexto de Análisis de Algoritmos y el comportamiento asintótico, Θ permite definir una relación de equivalencia. Indique formalmente cual es esa relación. Pruebe que efectivamente es una relación de equivalencia a partir de las *definiciones* dadas en el curso.
b) Partiendo de las *definiciones* dadas en el curso y siendo $f(n) = n^2 - 4n + 4$ y $g(n) = n^3 + 1$ probar que $f \notin \Theta(g)$

Nota: Para este ejercicio, si utiliza alguna propiedad debe demostrarla también a partir de las definiciones.

Ejercicio 4 (10 puntos)

Considere el algoritmo de ordenación *QuickSort*, del cual interesa que explique su funcionamiento. Para esto:

1. Indique qué técnica de diseño de algoritmos utiliza.
2. Detalle la estrategia que usa siguiendo esa técnica.
3. Escriba el algoritmo correspondiente en C* utilizando el siguiente algoritmo auxiliar:

```
void AUXILIAR(arreglo &a, int ini, int fin, int &pospiv){
  int i;
  int pivote;

  pivote = a[ini];
  pospiv = ini;
  for (i=ini+1; i<= fin; i++){
     if (a[i] < pivote){
        pospiv = pospiv +1
        Intercambiar(a, pospiv, i);
     }
  }
  Intercambiar(a, ini, pospiv);
}
```

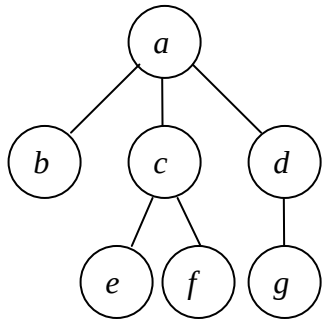
4. Calcule el costo del algoritmo de la parte 3. en su peor caso. Justifique detalladamente. ¿es óptimo en este caso?

Problemas

Problema 1 (30 puntos)

Dado un árbol (grafo) se requiere poder transformarlo de su formato estructurado (clásico) a un formato serial y viceversa. El formato estructurado consiste de vértices (con identificadores) y aristas entre vértices progenitores e hijos. El formato serial consiste en una secuencia ordenada de parejas, una para cada vértice, compuestas estas por el identificador y la cantidad de hijos. El orden en la secuencia queda establecido con los progenitores antecediendo a sus hijos, luego se toman ordenadamente los hijos como progenitores y se continúa sucesivamente de forma de alcanzar todos los vértices.

Los procedimientos de transformación son *serializar*, que realiza la conversión del formato estructurado al serial, y *des-serializar*, que establece la conversión inversa. A modo de ejemplo se tiene el siguiente grafo en sus formatos estructurado y serial:



Serializar →

← Des-serializar

a	3
b	0
c	2
d	1
e	0
f	0
g	0

Se pide:

a) Describir como se podrían modelar los procesos de serializar y des-serializar utilizando BFS.

b) Escribir un algoritmo en C* que a partir de un grafo en formato estructurado y su vértice raíz retorne su formato serial, según lo descrito en la parte a**)**:

```
void * Serializar_Arbol(Grafo * g, int raiz, Serial * s),
```

c) Escribir un algoritmo en C* que a partir de un grafo en formato serial retorne su formato estructurado y su vértice raíz, según lo descrito en la parte a**)**:

```
void * Desserializar_Arbol(Serial * s, Grafo * g, int * raiz).
```

Asumir que:

1. El grafo contiene *n* vértices que se identifican con números de 0 a *n*–1,
2. El formato estructurado se almacena en un TAD Grafo de enteros, `Grafo * g`, el cual no es necesario especificar.
3. El formato serial se almacena en un arreglo de parejas (identificador, cantidad de hijos), `Serial * s`, según lo descrito anteriormente. En el caso de utilizar TAD auxiliares, solo declarar sus estructuras y operaciones.

Problema 2 (30 puntos)

A un programador le ofrecieron implementar ***n*** módulos, los cuales se identifican con números de ***1*** a ***n***. Cada módulo i, $1 \leq i \leq n$:

- lleva t_i días de implementación, siendo t_i un entero mayor que cero.
- tiene una fecha límite de entrega v_i.
- genera p_i pesos de ingreso.

Se dice que un subconjunto S de {***1***, …, ***n***} es ***válido*** si es posible implementar en algún orden todos los módulos pertenecientes a S de tal modo que se cumplan las fechas límites de todos esos módulos.
El programador debe decidir que subconjunto válido de {***1***,…,***n***} implementa para obtener el máximo ingreso.

Aclaraciones:

i) el problema consiste en elegir los módulos a implementar, no el orden en que deben implementarse.
ii) las fechas límite están dadas en días y el programador comienza a trabajar el día 1.

Ejemplo: suponga que los módulos propuestos son los siguientes y que cada uno se implementa en un día:

Módulo	**Precio**	**Fecha Límite**
1	30	3
2	20	1
3	40	1
4	20	2

El programador puede seleccionar el subconjunto {***1***, ***2***}. Notar que no puede implementar ***1*** el primer día y ***2*** el segundo porque este último módulo no lo entregaría a tiempo. Sin embargo si implementara ***2*** el primer día y ***1*** el segundo día cumpliría con las fechas límite de ambos módulos. Por lo tanto este subconjunto es válido. El ingreso obtenido sería $50.
También es válido el subconjunto {***1***, ***2***, ***4***} porque se pueden implementar en el orden ***2***, ***4***, ***1***, en este caso el ingreso sería $70.
El ingreso máximo que puede obtener es $90 si selecciona el subconjunto {***1***, ***3***, ***4***}.
No son válidos los subconjuntos que incluyen a la vez a ***2*** y a ***3***.

Se pide:

a) El programador construirá un conjunto válido S, con los módulos que aceptará implementar usando el siguiente procedimiento:

```
Inicialmente S está vacío.
Se considera cada módulo i en orden no creciente de los cocientes pi/ti (precio por día de trabajo)
Si, junto con los módulos que ya haya aceptado, obtiene un subconjunto válido entonces agrega el módulo i a los aceptados. O sea, si S⋃{i} es un subconjunto válido, entonces incluye i en S.
```

Explique porqué se obtiene una solución óptima o muestre un ejemplo en el que la solución obtenida no es la que brinda el ingreso máximo.

Nota: Recuerde que, de acuerdo a la definición de conjunto válido, el orden en que los módulos son aceptados no tiene por que ser el mismo en que van a ser implementados.

b) Resolver el problema usando la técnica de **Backtracking**. Formalice el problema utilizando una tupla de largo variable. Describa formalmente y en lenguaje natural forma de la tupla, restricciones explícitas e implícitas y, si corresponde, función objetivo y predicados de poda.