Examen de Programación 3 y III (12/02/2010)

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **4** carillas. El total de puntos es 100. Para su aprobación necesita 60 puntos.
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* y el uso de *cout* y *cin*.
3. NO se puede utilizar ningún tipo de material de consulta. Puede usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.
4. No se contestaran 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 sólo 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 Teórica Obligatoria

Esta parte es eliminatoria, vale **40** puntos y usted debe obtener como mínimo el **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)

1. Dado el siguiente algoritmo:

```
void func (int* Vector, int L){
       for ( int i=0; i < L; i++){
              Vector[i]=0;
              for ( int j=i+1; j < L; j++){
                     float calc = random();//calcula numero real aleatorio entre
                                           //0 y 1, con distribución uniforme.
                     if( calc > 0,5){
                            Vector[j-1] = Vector[j] + Vector[j-1];
                     }
              }
              for( int k=0; k < L; k++){
                     cout << Vector[k] << endl;
              }
       }
}
```

Calcule el costo en los para los siguientes casos: mejor, peor y promedio. Considere como operación básica la **asignación** a un elemento de **Vector.**

2. Indique Verdadero o Falso **justificando** en cada caso. Justifique detalladamente sus respuestas demostrando o dando un contraejemplo partiendo de las **definiciones dadas en el teórico. Si utiliza alguna otra propiedad debe demostrarla.**

a. $\forall k \in R^+, \forall j \in R^+, k > j, \left(n^k \in O(n^j)\right)$

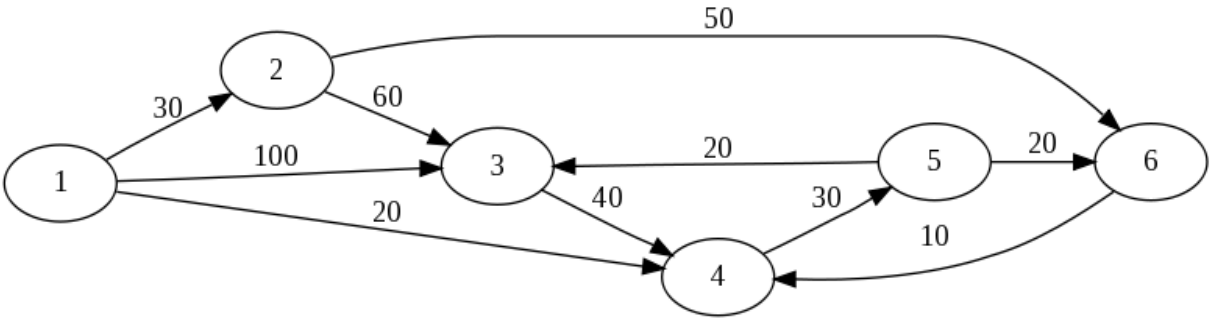
b. Si $f(n) \in O(n)$ y $g(n) \in \Theta(f(n))$ entonces $f(n) \times g(n) \in O(n^2)$

Ejercicio 2 (10 puntos)

1. Defina AVL e indique los órdenes de inserción y búsqueda para el caso promedio.
2. Dado un AVL vacío, y la siguiente lista de elementos: 60, 45, 30, 18, 39, 36, 90, 54, 42, 120, 51; haga un dibujo que muestre dicha estructura para cada inserción y cada rotación que aplique, indicando cual es la rotación que se aplica.

Ejercicio 3 (10 puntos)

1. Explique cómo funciona el algoritmo de Dijkstra, a qué técnica pertenece.
2. Aplique Dijkstra al vértice 1 en el grafo de la figura. Muestre los costos y predecesores de cada nodo para cada iteración del algoritmo.



Ejercicio 4 (10 puntos)

1. Defina Árbol de Decisión e indique que representan sus componentes.
2. Muestre como queda el árbol de decisión para el siguiente algoritmo de ordenación con T=[A,B,C], n=3.

```
void ordenar(int *T, int n) {
	for (int i=1; i < n ; i++) {
		int x = T[i];
		int j = i - 1;
		while (j >= 0 && x < T[j]) {
			T[j+1] = T[j];
			j--;
		}
		T[j+1] = x;
	}
}
```

Problemas

Problema 1 (30 puntos)

Sea un grafo $G = (V, A)$ dirigido y sin lazos, conteniendo n vértices que se identifican con números de *0* a n-1. G está representado mediante una matriz de adyacencia $n \times n$ con n constante.

En una recorrida DFS de G, las aristas de A se pueden clasificar en

- ***árbol*** (tree)**:** aristas a vértices no marcados en la recorrida, forman el árbol (o bosque)
- ***regreso*** (back): aristas de un vértice a un ancestro en un árbol de la recorrida
- ***directa*** (forward): aristas de un vértice a un descendiente en un árbol de la recorrida
- ***cruzada*** (cross): aristas entre ramas del árbol

En una recorrida DFS de G, para todo vértice v de V, sean c_v y f_v los tiempos de comienzo y fin de su procesamiento. Dados los vértices v y w, suponiendo que el vértice v es visitado antes que el vértice w, es decir $c_v < c_w$, se cumplen

- $c_w < f_v \Rightarrow c_v < c_w < f_w < f_v$
- $f_v < c_w \Rightarrow c_v < f_v < c_w < f_w$

Describa e implemente en C* un algoritmo DFS sobre G que, mediante la utilización de los tiempos de procesamiento, retorne la clasificación de cada una de sus aristas según el recorrido: *árbol*, *regreso*, *directa* o *cruzada*. El algoritmo deberá retornar una matriz M, análoga a la representación de G, tal que:

- $M[i, j] = 0 \Leftrightarrow (i, j) \notin A$
- $M[i, j] = 1 \Leftrightarrow (i, j)$ es arista *árbol*
- $M[i, j] = 2 \Leftrightarrow (i, j)$ es arista *regreso*
- $M[i, j] = 3 \Leftrightarrow (i, j)$ es arista *directa*
- $M[i, j] = 4 \Leftrightarrow (i, j)$ es arista *cruzada*.

Notas:

i. Asuma como dado el grafo G original, cualquier otra estructura que se utilice deberá ser definida y cargada como corresponda; también, implemente cualquier función auxiliar que utilice.
ii. Utilice la representación de matriz de adyacencia del grafo G directamente.

Problema 2 (30 puntos)

Un productor agrícola posee un campo para cultivo, dividido en *K* sectores. Al comienzo del período de cultivo, el productor debe decidir que cultivos realizar en cada sector. Existen *M* tipos diferentes de semillas para cultivar.
De cada semilla (s) se conoce el costo para cultivar un sector (c_s), el precio de venta de la cosecha de un sector (v_s) y el tiempo necesario para el cultivo de un sector (h_s).
El productor puede trabajar hasta *H* horas.
Se desea conocer cuales cultivos realizar en los distintos sectores de manera que le genere al productor la mayor ganancia posible, tal que el costo de realizar los cultivos no supere el presupuesto establecido *P* y además no se superen las *H* horas de trabajo.

Observación:

Solo se puede cultivar un tipo de semilla por sector y no hay dos sectores que cultiven la misma semilla.
Pueden existir sectores sin cultivar.

Se pide:

(15 puntos) Formalizar el problema en términos de Backtracking. Indicar: forma de la solución, restricciones explícitas e implícitas, función objetivo y predicados de poda si corresponden. Para esto utilizar lenguaje formal, y a su vez indicar la semántica de cada restricción en lenguaje natural.

(15 puntos) Implementar la función *maxGanancia* en C*:

```
Solucion* maxGanancia(int M, int K, int P, int H, int* costoSemilla, int* horasSemilla,
        int* precioVentaSemilla);
```

que retorna una solución de máxima ganancia. Los parámetros de la función son los siguientes:

- *M*, es la cantidad de semillas para cultivar.
- *K,* es la cantidad de sectores de cultivo.
- *P,* es el presupuesto disponible para cultivar.
- *H*, es la cantidad de horas de trabajo disponible.
- *costoSemilla*, contiene el costo de cultivar un sector, para cada tipo de semilla.
- *horasSemilla,* contiene la cantidad de horas necesarias para cultivar un sector, para cada tipo de semilla.
- *precioVentaSemilla*, contiene el precio de venta de la cosecha de un sector, para cada tipo de semilla.

```
struct Solucion
{
        int* tupla;//contiene los elementos de la tupla solución
        int ganancia;//ganancia total de la solución para el productor
        int cultivos;//cantidad de cultivos realizados de la solución
        int costo;//costo total de los cultivos de la solución
        int  horas;//cantidad de horas necesarias para realizar los cultivos de la solución
};
Solucion* crearSolucion(int tamTupla);
//crea una solución vacía, todos los enteros inicializados a 0 y el arreglo tupla de tamaño tamTupla
void destruirSolucion(Solucion* &s);
//libera la memoria de la solución s
Solucion* copiarSolucion(Solucion* s);
//crea y retorna una copia de la solución s
```

NOTA: No se corregirá la parte b) si no se realizó satisfactoriamente la parte a)