Examen de Programación 3 y III (08/01/2014)

Instituto de Computación, Facultad de Ingeniería, UdelaR

i. Este examen dura 4 horas y contiene **4**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)

2. Observemos primero que toda función F cumple que si $k \in \mathbb{R}^+$,

$$kF \in O(F) \tag{1}$$

Efectivamente, $\lim_n \frac{k \times F(n)}{F(n)} = \lim_n k = k \in \mathbb{R}^+$, y concluimos por la regla del límite que $kF \in O(F)$.

(a) **Verdadera.** Tenemos que $f \in O(2h)$ (por hipótesis) y que $2h \in O(h)$ (por 1); concluímos por propiedad del teórico que $f \in O(h)$.

(b) **Verdadera.** Como $f \in O(2h)$ y $g \in O(h)$, entonces existen c_F, n_F, c_G, n_G tales que

$$(\forall n \in \mathbb{N} : n > n_F : f(n) \leq c_F \times 2 \times h(n))$$
$$(\forall n \in \mathbb{N} : n > n_G : g(n) \leq c_G \times g(n))$$

Luego,

$f + g \in O(3h)$
$\Longleftarrow$ (1)
$f + g \in O(h)$
$\Longleftarrow$ (Def.)
$(\exists c \in \mathbb{R}^+ :: (\exists n_0 \in \mathbb{N} :: (\forall n \in \mathbb{N} : n > n_0 : f(n) + g(n) \leq c \times h(n))))$
$\Longleftarrow$ (tomando $c = c_F * 2 + c_G, n_0 = \max\{n_F, n_G\}$)
$(\forall n \in \mathbb{N} : n > \max\{n_F, n_G\} : f(n) + g(n) \leq (c_F \times 2 + c_G) \times h(n))$
$\Longleftarrow$
$(\forall n \in \mathbb{N} : n > n_F : f(n) \leq c_F \times 2 \times h(n))$ y $(\forall n \in \mathbb{N} : n > n_G : g(n) \leq c_G \times h(n))$,

y esto último lo obtuvimos de las hipótesis.

(c) **Falsa.** Sean las funciones h, f, g que cumplen

$$(\forall n \in \mathbb{N} :: h(n) = n, f(n) = n, g(n) = 1)$$

Vamos a mostrar que $f \in O(2h)$, $g \in O(h)$, y $\frac{f}{g} \notin O(2)$.

$f \in O(2h)$ Quiero justificar que $n \in O(2n)$; inmediato por 1.

$g \in O(h)$ Quiero justificar que $1 \in O(n)$; inmediato a partir de la regla del límite porque $\lim_n \frac{1}{n} = 0$.

$\frac{f}{g} \notin O(2)$ Quiero justificar que $n \notin O(2)$; inmediato a partir de la regla del límite porque $\lim_n \frac{2}{n} = 0$.

Ejercicio 2 (10 puntos)

1. Verdadero. El árbol T generado en el DFS es un camino simple, y el grafo cuenta con una única arista back que llega a la raíz y no aparece en T. Desde todo nodo se puede acceder a la raíz recorriendo ese camino y tomando finalmente esa única arista back. Por lo tanto, en todo nodo la función masalto toma el valor de prenum en la raíz, que es uno.

2. Falso. Considere el grafo G con cinco vértices, v;w1;w2; u1; u2, y seis arcos vw1;w1w2;w2v; vu1; u1u2; u2w, y considere que T tiene a v por raíz. El etiquetado del algoritmo marca todos los nodos con maasalto en uno, ya que la raíz está etiquetada con uno y de todos los restantes nodos se puede volver a la raíz usando una sola arista back. Sin embargo, G no es un ciclo.

Ejercicio 3 (10 puntos)

bool Busqueda(arreglo A, int inicio, int fin, elemento x)

```
{
(1) if (inicio == fin)
{
        (2) return A[inicio] == x;
}
(3) else
{
        (4) int medio;
        (5) medio = (inicio + fin)/2;
        (6) return Busqueda(A, inicio, medio, x) || Busqueda(A, medio+1, fin, x);
}
}
```

El algoritmo se invoca: bool pertenece = Busqueda(A, 0, n-1, x);

Los pasos claves de la técnica son: descomposición del problema (5 y 6, dividiendo el rango de búsqueda a la midad) y composición de soluciones (6, con el operador ||).

Por otro lado, se arriba a un problema cuya solución es conocida (2, problema que no requiere ser descompuesto) cuando se verifica la condición del paso base (1).

Ejercicio 4 (10 puntos)

La estrategia óptima es la b).

Para justificarlo se podrían utilizar básicamente dos estrategias:

1.- demostrar que b) es óptima (ver notas del teórico)

2.- mostrar, con un contra ejemplo, que a) no es óptima, de lo cual se desprende que b) debe ser óptima ya que no hay más opciones.

Estrategia de justificación 2 : contra ejemplo.

Alcanza con mostrar que la estrategia b) es mejor que la a) para una instancia, convenientemente elegida, del problema.

Sea M=n y los objetos i:p=10, w=n; j:p=6, w=n/2 y k:p=6, w=n/2.

Según a)

- Se elegiría primero el objeto i por ser el de mayor ganancia p=10.
- No se elegiría ningún otro objeto porque con una unidad de i se completa la capacidad de la mochila.
- Ganancia total=10.

Según b)

- Se elegiría primero el objeto j por ser pj/wj >= pk/wk >= pi/wi.
- Continuando en orden decreciente de p/w se eligiría el objeto k.
- No se elegiría ningún otro objeto porque con una unidad de j y otra de k se completa la capacidad de la mochila.
- Ganancia total=12.

Entonces a) no es óptima.

```
void KnapSock (float M, int n, float *solucion, float *W)

{

    int i;

    float capacidad;

    for(i = 0; i < n; i++)

        solucion[i] = 0;



        capacidad = M;

        i = 0;

        while ((i < n) && (W[i] <= capacidad))

        {

            solucion[i] = 1;

            capacidad = capacidad - W[i];

            i++;

        }

    if (i < n)
```

```
        solucion[i] = capacidad / W[i];

    }
```

Problemas

Problema 1 (30 puntos)

Restricciones Explícitas:

- Hay 8 grupos en total, N=8.
- Cada grupo tiene 4 equipos, M=4.
- Cada país se representa con un entero de 1 a P.
    - Siendo $T = < t_1, \dots, t_i, \dots, t_N >$ y cada t_i de la forma $t_i = < x_1, \dots, x_j, \dots, x_M >$ se cumple que $\forall i \in \{1..N\},\ \forall j \in \{1..M\}: x_{ij} \in \{1..P\}$.
- Cada grupo está formado por un país de cada bolillero.
    - Siendo $t_i = < x_1, \dots, x_4 >$ el grupo i, se cumple que $x_j \in bol_j\ \forall\ j = 1,..4, \forall\ i = 1,..8$

- Uruguay clasifica primero o segundo en su grupo.
  El primer clasificado de un grupo es el que gana los 3 partidos, es decir, todos los partidos que juega.
  En caso de que no haya empate en el segundo puesto, clasifica en segundo lugar el país que haya ganado 2 partidos. En caso de empate, es decir, que hay 3 países que ganaron un sólo partido, clasifica el país con mejor ranking entre los empatados.

    - Si $1 \in t_i \ \rightarrow 1\ \in \{primero(t_i),\ segundo(t_i)\}$.
      Siendo
      $a)\ primero(G) = pais_k\ \in G/1 = ganador(pais_k,\ pais_n)\ \forall\ pais_n \in G,\ pais_n \neq pais_k.$
      $b)\ segundo(G) = pais_k\ \in G/2 = \sum_{\substack{pais_n \in G \\ pais_n \neq pais_k}} ganador(pais_k, pais_n)$

      $\vee$

      $\#E = 3,$

      siendo $E = \left\{ pais_h \in G / \sum_{\substack{pais_n \in G \\ pais_n \neq pais_h}} ganador(pais_h, pais_n) = 1 \right\} \wedge \quad pais_k.ranking = \min\{pais_h.ranking\}\ \forall\ pais_h \in E$

      c) $ganador(A, B) = A.PGHistorial(B) > B.PGHistorial(A)$, esta función retorna 1 en caso de desigualdad y 0 en caso contrario.

Restricciones implícitas:

- Los países no se repiten.
    - $\forall\ i, m\ \in \{1..N\},\ \forall\ j, n\ \in \{1..M\} : si\ i \neq m\ v\ j \neq n\ \rightarrow\ x_{ij} \neq x_{mn}$

Función Objetivo:

- No hay.

Predicado de Poda:

- No hay.

Problema 2 (30 puntos)

```
ListaEnteros seleccionDeCajeros(Grafo g, int** dist_km, int R ) {
	// inicialización de estrucutras utilizadas
	int N = cantidadNodosGrafo(g);
	bool* visitados = new bool[N];
	int* vectorDistancias = new int[N];
	for (int i = 0 ; i< N ; i++) {
		visitados[i] = false;
		vectorDistancias[i] = 0;
	}
	ListaEnteros listaSolucion = crearListaEnteros();
	ListaEnteros colaProcesamiento = crearListaEnteros();
	// el bfs parte desde el banco
	agregarFinal(colaProcesamiento, 0);

	while(!esVacia(colaProcesamiento)) {
		int nodoActual = sacarPrimero(colaProcesamiento);
		int distanciaNodoActual = vectorDistancias[nodoActual];
		ListaEnteros adyacentes = obtenerAdyacentes(g, nodoActual);
		while(!esVacia(adyacentes)) {
			int nodoAdyacente = sacarPrimero(adyacentes);
			if (visitados[nodoAdyacente] == false) {
				agregarFinal(colaProcesamiento,nodoAdyacente);
				visitados[nodoAdyacente] = true;
				vectorDistancias[nodoAdyacente] = distanciaNodoActual + 1;
			}
		}
		// si el nodo se encuentra en zona de color
		if (distanciaNodoActual mod 2 == 1) {
			bool yaExisteCercano = false;
			// recorro todos los nodos pertenecientes hasta el momento a la  solucion en busca de un nodo de la misma
zona del nodo actual, que además diste de él menos de R km
			iniciarRecorrida(listaSolucion);
			int nodoSolucion;
			while (siguienteEn(listaSolucion,&nodoSolucion) && !yaExisteCercano) {
				if (vectorDistancia[nodoSolucion] == distanciaNodoActual &&
dist_km[nodoSolucion,nodoActual] < R) {
					yaExisteCercano = true;
				}
			}
			// si no existe ningún otro nodo cercano en la solución, entonces lo agrego a la misma
			if (!yaExisteCercano) {
				agregarFinal(listaSolucion,nodoActual);
			}
		}
	}
	return listaSolucion;
}
```