Primera Convocatoria Estructuras de Datos y Algoritmos II.
2º Grado en Ingeniería informática 6 de Junio de 2014

TEORIA

1. El siguiente pseudocódigo muestra la implementación de un algoritmo que se aplica a un Montículo Binario.

```
función algoritmoX(ref m: tipoMonticulo): lógico
1     i, j: entero
2     i ← 1
3     mientras (i <m.tamaño div 2) hacer
4         j ← 2 * i
5         si j ≠ m.tamaño entonces
6             si m.elemento[j+1].clave<m.elemento[j].clave entonces
7                 j ← j + 1
8             fin si
9         fin si
10        si m.elemento[i].clave < m.elemento[j].clave entonces
11            i ← i +1
12        sino
13            devolver FALSO
14        fin si
15    fin mientras
16    devolver VERDADERO
```

a) ¿Qué resultado obtiene?

b) Explicar brevemente en qué estrategia se basa

c) Analizar brevemente su comportamiento en función del número de elementos del montículo N.

2. Teniendo en cuenta el siguiente listado, que muestra en pseudocodigo, la implementación del algoritmo de PRIM, responde breve y razonadamente a las siguientes cuestiones:

```
procedimiento PRIM(vInicio: tipoldVértice, ref g: tipoGrafo)
 1      p : punteroArco
 2      v,w : tipoldVértice
 3      i: entero
 4      iniciar(g)
 5      g.directorio[vInicio].peso ← 0
 6      g.directorio[vInicio].anterior ← 0
 7      para i=l hasta g.orden hacer
 8          v ← buscarVérticePesoMinimoNoAlcanzado(g)
 9          g.directorio[v].alcanzado ← VERDADERO
10          p ← g.directorio[v].listaAdyacencia
11          mientras p ≠ NULO hacer
12              w ← p↑.vértice
13              si not(g.directorio[w].alcanzado) entonces
14                  si g.directorio[w] .peso > p↑.peso entonces
15                      g.directorio[w].peso ← p↑.peso
16                      g.directorio[w].anterior ← v
17                  fin si
18              fin si
19              p ← pt↑.sig
20          fin mientras
21      fin para
```

(4 puntos)

a) Analizar su comportamiento

b) Indicar como puede mejorarse analizando la mejora obtenida.

c) Dada una variable vGrafo de tipoGrafo, escribir el pseudocodigo que permita, utilizando el resultado del algoritmo de Prim, obtener el grafo que representa el árbol de expansión de vGrafo.

3. Utilizando la tabla de la figura estudiar los casos que deben tenerse en cuenta en el algoritmo de inserción de un nodo en un árbol balanceado cuando se regresa por el camino de búsqueda, después de la inserción de un nodo por la rama **derecha**. Si en algún caso es necesaria la reestructuración del subárbol, indicar qué tipos de rotaciones son necesarias y de qué depende el tipo de rotación.

(3 puntos)

Antes de la INSERCION			Después de la INSERCION		
nodo↑.fe	subárbol	altura	subárbol	nodo↑ .fe	altura

Explica razonadamente por qué en los algoritmos de inserción y eliminación de árboles balanceados se utiliza un parámetro de tipo lógico que indica si la altura del árbol ha cambiado.

4. En una organización secuencial indexada se ha recurrido a la técnica de área de desborde para almacenar los nuevos registros que se inserten después de su creación inicial. Despues de varios meses se tienen bastantes registros desbordados y se observa penalización en el rendimiento de acceso al fichero. Razona brevemente a qué es debida esta penalización y cómo debería resolverse el problema.

(1 punto)

Primera Convocatoria Estructuras de Datos y Algoritmos II.
2º Grado en Ingeniería informática 6 de Junio de 2014

PRACTICA

1. Se define la profundidad de un nodo en un árbol como el número de aristas en el camino único que permite acceder a él desde la raíz.

a) Teniendo en cuenta que la raíz tiene profundidad 0, codificar en C una función recursiva que calcule la profundidad de todos los nodos de un árbol binario, según el siguiente prototipo.

```
void profundidad( Arbol a, int profNivelAnt);
```

Utilizar para la implementación los tipos que se proporcionan a continuación:

```
typedef char tipoInfo;
typedef struct tipoNodo {
  tipoInfo Info;
  int profundidad;
  struct tipoNodo *izq, *der;
} tipoNodo;

typedef tipoNodo * Arbol;
```

b) La función anterior no está encapsulada en el sentido de que su correcto funcionamiento depende del valor del segundo parámetro en la primera invocación de la función. ¿Cuál es el valor correcto en su invocación? ¿Cómo podríamos conseguir encapsularla y hacerla representativa del Tipo Abstracto Arbol, de forma que su correcto funcionamiento no dependa del usuario del TAD?

2. Implementar en C las tres operaciones básicas que de definen para el TAD Conjuntos Disjuntos. Previamente a la implementación debe elegirse un nivel de representación y definir los tipos y prototipos adecuados para poder implementar las operaciones. Para ello ir respondiendo a cada una de las siguientes cuestiones:

a) Indicar **claramente** la representación elegida en base a la cual se realizaran siguientes cuestiones.

b) Definición de tipos y prototipos de las funciones que implementaran las operaciones básicas.

c) Implementación de las funciones

3. Dado un archivo (asignaturas.hash) organizado mediante el Método de Dispersión según los siguientes criterios:
   - El fichero está organizado en cubos con **CAPACIDAD** para **20** registros de tipoAsignatura cada uno.
   - Se han considerado **50 CUBOS** inicialmente y **10 CUBOS DE DESBORDE** más para almacenar los registros asignados a cubos LLENOS.
   - Cada cubo almacena además el número de registros que se le han asignado, incluyendo los que no haya podido almacenar porque estaba lleno.
   - Los registros desbordados se han almacenado **secuencialmente** en el área de DESBORDE. Los 20 primeros registros desbordados en el primer cubo de desborde, los 20 siguientes al siguiente cubo de desborde, etc.
   - Se ha utilizado como clave el campo CODIGO de los registros tipoAsignatura y la función **módulo** como función hash para asignar un registro a un cubo.

Los primeros tres criterios se reflejan en las definiciones que se proporcionan a continuación:

```
#define C 20                //  Capacidad del cubo
#define CUBOS 50            //  Número de cubos en el a rea prima
#define CUB0SDESB0RDE 10    //  Número de cubos area de desborde

typedef struct {
  int codigo;
  char nombre[20];
  char titulación[20];
  int créditos;
  char curso;
} tipoAsignatura;

typedef struct {
  tipoAsignatura reg[C];
  int numRegAsignados;
} tipoCubo;
```

Implementar una función:

```
int expansionTotal (char * ficheroEntrada, char * ficheroSalida);
```

que modifique el fichero hash de entrada que se le pasa como primer parámetro mediante una expansión total, es decir, duplicando el número de cubos inicialmente asignados y también los cubos de desborde. La función debe devolver el número de registros desbordados después de efectuar la expansión.