*Carnet* _____ *Nombre Completo* _____

# CI-2126 Computación II

Examen II (30%) Sep/Dic 2010

Lea detenidamente todas las preguntas del examen antes de empezar a contestar.

1) *(1 pts)* Defina los términos *raíz* y *hoja* en el TDA Árbol:

2) *(1 pts)* ¿Cual es la diferencia entre altura y nivel en un árbol ?:

3) *(7 pts)* Indique con **V**(erdadero) o **F**(also) a las expresiones siguientes:

.- El algoritmo de ordenamiento *Burbuja* (*Bubblesort*) está entre los más rápidos. F

.- Los punteros ocupan espacio en memoria sólo cuando no son nulos. F

.- Un Árbol que sólo permite un hijo como máximo es una Lista. V

.- Ordenamiento por *Fusión* es igual al de *Inserción*, pero de atrás para adelante F

.- La cola es una estructura del tipo FIFO (First In First Out). V

.- La pila es una estructura del tipo FIFO (First In First Out). F

.- Dos pilas encadenadas, una construida a partir de vaciar la otra, equivalen a otra Pila F

.- La altura máxima de un árbol binario con $n$ elementos es $2^n$ F

.- Si el recorrido un árbol en *Preorden* se invierte se obtiene el de *Postorden* F

.- Una raíz de un árbol puede ser también una hoja. V

.- Un puntero a función no se puede pasar como parámetro a otras funciones. F

.- En árboles binarios todo nodo/hoja tiene estrictamente 2 hijos o menos V

.- El TDA *ListaDobleCircular* se puede usar tanto para Pila como para Cola V

.- Sea B el árbol resultante de permutar los hijos derechos e izquierdo del árbol A en forma recursiva. El recorrido *Inorden* de ambos es igual. F

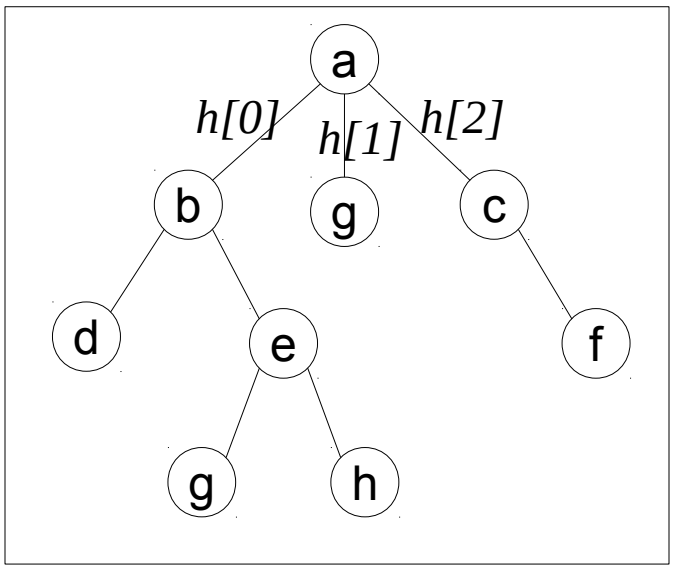
4) Para la definición de el TDA Árbol *n-ario*, tenemos la siguiente definición y constructor

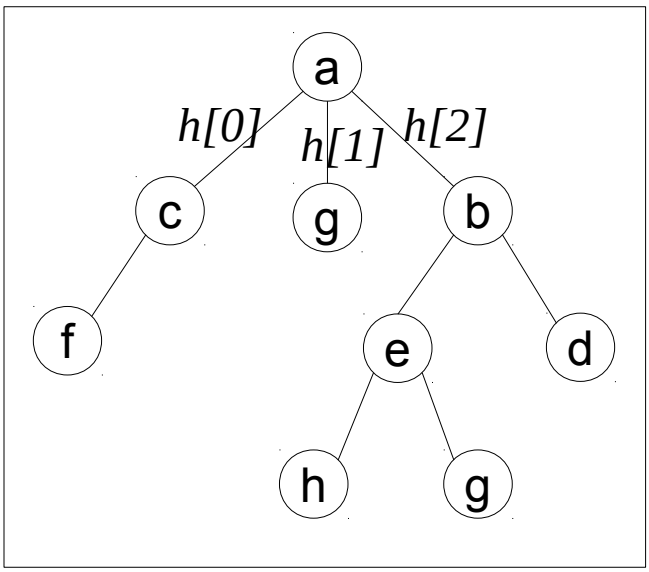
```
typedef struct Nodo* Arbol;
typedef struct Nodo {
   Elem  info;
   int   numHijos;
   Arbol hijo [DIM];   /* La constante  DIM es estrictamente positiva, DIM >= 2 */
} Nodo;

   /* Constructor de un Árbol raíz con un Elem y sus hijos */
Arbol cons(Elem e, int N, Arbol* h) {
   Arbol R = malloc(sizeof(Nodo));
   for (int k = 0; k < N; k++) {
      R->info    = e;
      R->hijo[k] = h[k];
   }
   return R;
}

   /* Devuelve un Arbol como transformación del Árbol A */
Arbol acertijo(Arbol A) {
   if (!A) {
      return NULL;
   } else {
      Arbol aux[DIM];
      int N = A->numHijos;
      for (int k = 0; k < N; k++) {
        aux[k] = acertijo( A->hijo[N-k-1] );
      }
      return cons(A->info, N, aux);
   }
}
```

¿ (5 puntos) Cual es la estructura de W si hacemos a W = *acertijo*(T), con T el árbol de abajo?




¿ (1 punto) Qué cree usted que hace la función *acertijo*() y que sugiere como su verdadero nombre ?

La función intercambia los hijos con respecto al centro del nodo (derecha con izquierda).

Podemos llamar a al función *espejo*, *flip*, *reflejar*

5) (5 ptos) El algoritmo de Búsqueda Binaria visto en clase, int *BB(A, li, ls, clave)* recibe un arreglo A *ordenado*, una *clave* a buscar, un subíndice inferior del arreglo (*li*) y un subíndice superior del arreglo (*ls*). En cada iteración, obtiene el subíndice medio del arreglo (*lm = (li + ls) / 2*). Si el valor es menor que A[lm], hace recursión entre los límites *li* y *lm-1*. Si es mayor, hace recursión entre los límites *lm+1* y *ls*. Si es igual, devuelve el subíndice *lm* como valor de retorno (>=0). Si *li > ls*, significa que no se encuentra el valor en el arreglo, y devuelve **-1**.

Codifique una función recursiva llamada Búsqueda Cuaternaria, *int BC(A, li, ls, clave)*, que en vez de un punto medio, calcule tres puntos equidistantes de referencia:

```
la = (li+ls)/4          lb = 2*(li+ls)/4        lc =3*(li+ls)/4;
```

.- Si es igual a A[la], A[lb] o A[lc], devuelve *la, lb* o *lc* respectivamente

.- Si la clave es menor que A[la], hace recursión en el primer cuarto *(li, la-1)*

.- Si la clave es menor que A[lb], hace recursión en el segundo cuarto *(la, lb-1)*

.- Igual para el tercer y el último cuartos.

```
int BC (int* A, int li, int ls, int clave) {
   if (li > ls) return -1;
   la = 1*(li+ls)/4;
   lb = 2*(li+ls)/4;
   lc = 3*(li+ls)/4;
   if (clave <  A[la]) return BC(A, li, la-1, clave);
   if (clave == A[la]) return la;
   if (clave <  A[lb]) return BC(A, la, lb-1, clave);
   if (clave == A[lb]) return lb;
   if (clave <  A[lc]) return BC(A, lb, lc-1, clave);
   if (clave == A[lc]) return lc;
   return BC(A, lc, ls, clave);
}
```

6)(5 puntos) Codifique un algoritmo *sonIguales* que reciba dos listas planas y determine si ambas listas son iguales. En caso de ser iguales debe retornar **0**, si son distintas, debe retornar la posición en la lista en que dejaron de ser iguales.

```
sonIguales: Lista x Lista --> int

Por ejemplo, para las listas
A = 2->3->4,  B = 2->3->6->7,  C = 2->3->6->7 y D = null:

sonIguales(A, A) == 0,     sonIguales(A, D) == 1
sonIguales(A, B) == 3,     sonIguales(B, C) == 0;
```

Versión recursiva:

```
int sonIguales_aux (int cont, Lista A, Lista B) {
   if ((A == NULL) and (B == NULL)) return 0;
   if ((A == NULL) or  (B == NULL)) return cont;
   if ((A->info != B->info)) return cont;
   else return sonIguales_aux(cont+1, A->siguiente, B->siguiente);
}

int sonIguales (Lista A, Lista B) {
   return sonIguales_aux(1, A, B);
}
```

Versión iterativa:

```
int sonIguales (Lista A, Lista B) {
   int cont = 1;
   while ((A != NULL) and (B != NULL)) {
      if (A->info != B->info)
         return cont;
      cont++;
      A = A->siguiente;
      B = B->siguiente;
   }
   if ((A == NULL) and (B == NULL)) return 0;
   if ((A == NULL) or  (B == NULL)) return cont;
}
```

7) (5 puntos) Rellene los espacios en blanco de la función *Encolar* de una lista simple circular que se usa como cola. Recuerde que en una lista circular el puntero de la lista apunta al "último" de la cola, mientras que el "primero" de la cola es el siguiente del "último", excepto en el caso cuando la lista circular está vacía (cuando hay un sólo elemento, éste es "primero" y "último" a la vez, es decir, su siguiente es él mismo).

```
typedef struct NODOSTRUCT* NODO;

typedef struct NODOSTRUCT {
   ELEM  info;
   NODO siguiente;
} NODOSTRUCT;

typedef struct LCIRCSTRUCT* LCIRC;

typedef struct LCIRCSTRUCT {
   NODO  ultimo;
} LCIRCSTRUCT;

LCIRC LC_Encolar( LCIRC L, ELEM e) {
      /* PRE:   L != NULL                          */
      /* POST: coloca el elemento e como último de la lista */

   NODO nodo = malloc(sizeof(NODOSTRUCT));
   nodo->info      =   e   ;
   nodo->siguiente =  NULL ;

   if (L->ultimo == NULL) { /* Caso base  */
      nodo->siguiente = nodo;
   }  else  {
      nodo->siguiente      = L->ultimo->siguiente;
      L->ultimo->siguiente =  nodo ;
   }
   L->ultimo = nodo;
   return  L ;
}
```