CI2126 Computación II, Ene/Mar 2014

Solución Examen II (30%)

1.- (7 puntos) Indique con **V**(erdadero) o **F**(also) a las expresiones siguientes:

a.- `typedef struct {int valor} Val_t` declara la variable "Val_t" _**F**_

b.- Los punteros ocupan espacio en memoria proporcional al espacio referenciado _**F**_

c.- La inserción de valores ordenados en árboles binarios tiene un peor caso de orden *o(n)* _**V**_

d.- Mergesort es de orden $o(n \log_2 n)$ en tiempo, y su peor caso también _**V**_

e.- Quicksort en promedio es $o(n \log_2 n)$ en tiempo, y su peor caso también _**F**_

f.- Quicksort no requiere de espacio adicional para guardar resultados intermedios _**V**_

g.- La inserción y remoción de valores ordenados en árboles es de orden $o(\log_2 n)$ _**V**_

h.- Una Cola puede ser usada como base para un TDA Lista _**V**_

i.- Un archivo binario de registros se puede leer completo de una vez _**V**_

j.- No se puede escribir información adicional a un archivo binario _**F**_

k.- map, filter y reduce se implementan igual en Listas y Arboles _**F**_

l.- De un árbol ordenado, se produce una lista ordenada si se recorre *preorden* _**F**_

m.- Si se elimina la *raíz* de un árbol ordenado, el recorrido *enorden* del árbol resultante es el mismo tanto si se elimina el valor mayor por el hijo izquierdo como si se elimina el menor valor por el hijo derecho *(justifique)* _**V**_

El recorrido *enorden* del arbol (ordenado) resultante escribe la secuencia de valores en forma ordenada. Reemplazar la raíz por el mayor valor en su hijo izquierdo o por su menor valor en el hijo derecho no altera el orden relativo de los nodos del árbol (hijo izq <= padre <= hijo derecho), por lo que el recorrido *enorden* una vez removida la raíz resulta en la misma secuencia.

2.- **(5 puntos)** Dada la estructura de Arbol Binario de Búsqueda (ordenado) siguiente, ejecute "en frío" anotando las salidas que imprime *printArbol*, suponiendo que se tiene el siguiente Árbol de la forma `( info hIzq hDer )`, construido en el programa principal:

```
a  =  ( 12 ( 3 1 ( 8 · 10 ) ) ( 19 15 · ) );
```

```
typedef int Elem;
typedef struct ArbolCDT_t* Arbol;
typedef struct ArbolCDT_t {
    Elem   info;
    Arbol* hIzq;
    Arbol* hDer;
} ArbolCDT_t;

Arbol consArbol(Elem e, Arbol hi, Arbol hd);

Arbol misterio(Arbol a) {
    if (a) {
         Arbol b = consArbol(a->info,
                             misterio(a->hDer),
                             misterio(a->hIzq));
         printArbol(b);
         return b;
    } else {
         printArbol(a);
         return a;
    }
}
```

```
void printAr_aux(Arbol a) {
    if (a)
        if ( esHoja(a) )
            printf("%i ", a->info);
        else {
            printf("( %i ", a->info);
            printAr_aux(a->hIzq);
            printAr_aux(a->hDer);
            printf(") ");
        }
    else
        printf("· ");
}

void printArbol(Arbol a) {
    if (a)
         printAr_aux(a);
    else
        printf("( · ) ");
    printf("\n");
}
```

(a) ¿Qué imprimiría, línea por línea, la llamada `Arbol c = misterio(a)` ?

(b) Construya visualmente los árboles **c** y **a**. ¿Qué puede decir que hace entonces la función *misterio*?

RESPUESTA

(a)

```
( · )
( · )
1
( · )
( · )
( · )
10
( 8 10 · )
( 3 ( 8 10 · ) 1 )
( · )
( · )
15
( · )
( 19 · 15 )
( 12 ( 19 · 15 ) ( 3 ( 8 10 · ) 1 ) )
```

(b) Construye el árbol espejo o reflexión, es decir, un árbol ordenado en el sentido inverso (si de menor a mayor, pasa a ser de mayor a menor y viceversa)

3.- **(5 puntos)** Usted vió el algoritmo Búsqueda Binaria (*BBin*) recursivo en clase, que busca en un arreglo de registros (ordenado por un atributo *clave)* un *valor* en la región entre los límites inferior (***Li***) y superior (***Ls***) de subíndices del arreglo, comparando el *valor* a buscar con la *clave* del registro situado al subíndice mitad (***Lm***) del arreglo, y reduciendo la búsqueda a la mitad inferior o superior de la correspondiente región.

Se le pide que modifique este algoritmo para crear y codificar la Búsqueda Ternaria (*BTer*), que usa DOS puntos de comparación, uno a un 1/3 de la región de subíndices (***L1t3***) y el otro a 2/3 de la región de subíndices (***L2t3***). La función deberá devolver el *subíndice* del *valor* buscado si éste se encuentra en el arreglo, y **-1** si no se consigue.

```
typedef struct Registro_t {
    long clave;
    data info;
}  Registro_t;
```

```
typedef Registro_t* Arreglo;
(Equiv. typedef Registro_t Arreglo[MAXVAL])

int BTer (Arreglo A, int valor, int Li, int Ls);
```

RESPUESTA

```
     /* Algoritmo original de Búsqueda binaria */
int BBin(Arreglo a, int valor, int Li, int Ls)
{
    if (Li < Ls)
    {
        int Lm = Li + (Ls - Li)/2;
        if (valor < a[Lm].clave)
            return BBin(a, valor, Li, Lm);
        else if (valor == a[Lm].clave)
            return Lm;
        else if (valor > a[Lm].clave)
            return BBin(a, valor, Lm + 1, Ls);
    }
    return -1;
}

     /* Algoritmo de Búsqueda Ternaria */
int BTer(Arreglo A, int valor, int Li, int Ls)
{
    if (Li < Ls)
    {
        int L1t3 = Li + 1*(Ls - Li)/3;
        int L2t3 = Li + 2*(Ls - Li)/3;
        if (valor < A[L1t3].clave)
            return BTer(A, valor, Li, L1t3);
        else if (valor == A[L1t3].clave)
            return L1t3;
        else if ( (valor > A[L1t3].clave) and (valor < A[L2t3].clave) )
            return BTer(A, valor, L1t3 + 1, L2t3);
        else if (valor == A[L2t3].clave)
            return L2t3;
        else if (valor > A[L2t3].clave)
            return BTer(A, valor, L2t3 + 1, Ls);
    }
    return -1;
}
```

4.- **(6 puntos)** Para el tipo de datos abstracto **Lista** especificado abajo, implemente el algoritmo *Mergesort* (*fusión*) para listas.

El algoritmo Lista mergesort(Lista r) es como sigue:

Si **r** es vacía,
devolver **r**
sino
Picar **r** por la mitad devolviendo las dos mitades como listas, **ri** y **rd**
si = mergesort(**ri**);
sd = mergesort(**rd**);
Fusionar **si** y **sd** en **s** *(por orden ascendente de clave)*
devolver **s**;

-

AYUDA: ***Mergesort*** para listas NO crea nodos nuevos, sólo los mueve de lista.

```
typedef int Elem;

typedef struct NodoCDT_t* Nodo;

typedef struct NodoCDT_t {
    Elem clave;
    Nodo siguiente;
}  NodoCDT_t;

typedef struct ListaCDT_t* Lista;

typedef struct ListaCDT_t {
    int  tam;
    Nodo primer;
    Nodo ultimo;
} ListaCDT_t;
```

```
   /* Crea una Lista vacía */
Lista consLista () {
    Lista v = CREATE(ListaCDT_t);
    v->tam = 0; // cambiará según se añada o quite
    v->primer = v->ultimo = NULL;
    return v;
}
   /* Devuelve valor del primer Elem de la lista */
Elem primerElem(Lista s);

   /* Inserta un nodo al final de la lista */
void insertarNodoFinalLista(Nodo n, Lista s);

   /* Saca el primer nodo de la lista */
Nodo removerPrimeroLista (Lista s);
```

RESPUESTA

```
void picar(Lista r, Lista ri, Lista rd) {
    _pre(r and (r->tam > 1));

    int mi = r->tam/2;     /* Cardinalidad de la mitad izquierda */
    int md = r->tam - mi;  /* Cardinalidad de la mitad derecha   */

    while (mi--)  /* copiamos la 1ra mitad de r en ri */
        insertarNodoFinalLista( removerPrimeroLista(r), ri );

    while (md--)  /* copiamos la 2da mitad de r en rd */
        insertarNodoFinalLista( removerPrimeroLista(r), rd );

    /* r queda vacío */
}


Lista fusionar(Lista si, Lista sd, Lista s) {
    _pre(r and si and sd);

         /* Mientras ambas listas tengan elementos, mover ordenado a s */
    while ( !si->primer and !sd->primer ) {
        if ( primerElem(si) < primerElem(sd) )
            insertarNodoFinalLista( removerPrimeroLista(si), s );
        else
            insertarNodoFinalLista( removerPrimeroLista(sd), s );
    }
         /* Mover remanentes de si */
    while (si->primer)
        insertarNodoFinalLista( removerPrimeroLista(si), s );
         /* Mover remanentes de sd */
    while (sd->primer)
        insertarNodoFinalLista( removerPrimeroLista(sd), s );

    /* sd y si quedan vacíos */
    return s;
}


Lista mergesort(Lista r) {
    _pre(r);

    if (r and (r->tam > 1)) {
        Lista mizq = consListaVacia();
        Lista mder = consListaVacia();

        picar(r, mizq, mder);
        mizq = mergesort(mizq);
        mder = mergesort(mder);
        fusionar(mizq, mder, r);
    }
    return r;
}
```

5.- **(7 puntos)** A usted se le ha encargado diseñar la estructura de datos para poder implementar el manejo de resultados del Mundial FIFA 2014. Participan 32 equipos [0..31], asignados en 8 grupos [0..7] de 4 equipos cada uno, por el procedimiento asignar.

- La información original de cada equipo consiste en su código de equipo y el grupo al que fue asignado, más los partidos ganados, empatados o perdidos, y la diferencia de goles.
- Hay una primera ronda clasificatoria, que produce un ranking en cada grupo [0 .. 3] con 0 el más alto.

De esta primera ronda se crea la información para la ronda de eliminación subsiguiente.
- El primero del grupo 2k jugará contra el segundo del grupo 2k+1, y el primero del grupo 2k+1 jugará contra el segundo del grupo 2k.
- Cada partido jugado en esta segunda etapa tendrá un sólo equipo ganador, la estructura de cada partido debe tener el resultado del ganador del juego entre ambos. Si el valor es -1, el partido no se ha jugado.

(a) Diseñe la estructura de datos apropiada para la primera ronda, **EstR1**

(b) Diseñe la estructura de datos apropiada para la segunda ronda, **EstR2**

(c) Diseñe una función *construir*: **EstR1** --> **EstR2** que construya la base de la **EstR2**

~~(d) Diseñe una función *cambiar*: **EstR2** --> **EstR2** que recorra toda la **EstR2,** y que en base a averiguar los dos ganadores de partidos en la jerarquía rellene la información necesaria en el subsiguiente partido de esa segunda ronda.~~ **ELIMINADA**

RESPUESTA

(a)

```
typedef struct Equipo {
  char codigo;  // código de equipo
  char grupo;   // código de grupo
  char gan;     // juegos ganados
  char emp;     // juegos empatados
  char per;     // juegos perdidos
  char dif;     // diferencia goles
} Equipo;

Equipo equipo[MAXEQUIPOS]; // Son 32


typedef struct EstR1 {
  char grupo;       // código de grupo
  Equipo equipo[4]; // código de grupo
  char ranking [4]; // código de equipo
} EstR1;

EstR1 grupos[MAXGRUPOS];     // Son 8
```

(b)

```
typedef struct PartidoCDT* EstR2;

typedef struct PartidoCDT {
  char equip[2]; // cód. equipo
  char ganador;  // cód. equipo o -1
  EstR2 anterior[2]
  // Partidos anteriores
}  PartidoCDT;
```

(c)

```
EstR2 construir (EstR1* gr) {
        /* Equivalente a "PartidoCDT *
         * partidos[MAXGRUPOS*2]"    */
    EstR2 partidos = calloc(MAXGRUPOS*2, PartidoCDT);
    int k = MAXGRUPOS;

    for (k=0; k< MAXGRUPOS; k++) {
       partidos[  2*k  ].equip[0] = gr[  k  ].ranking[0];
       partidos[  2*k  ].equip[1] = gr[k + 1].ranking[1];
       partidos[2*k + 1].equip[0] = gr[  k  ].ranking[1];
       partidos[2*k + 1].equip[1] = gr[k + 1].ranking[0];
    }
    return partidos; /* Devuelve apuntador a partidos */
}
```