PROGRAMACIÓN II

Comenzado el	lunes, 18 de mayo de 2020, 11:00
Estado	Finalizado
Finalizado en	lunes, 18 de mayo de 2020, 12:01
Tiempo empleado	1 hora

Pregunta 1

Finalizado

Puntúa como 1,00

En los siguientes algoritmos, indica si cada una de las llamadas recursivas es de cola o no:

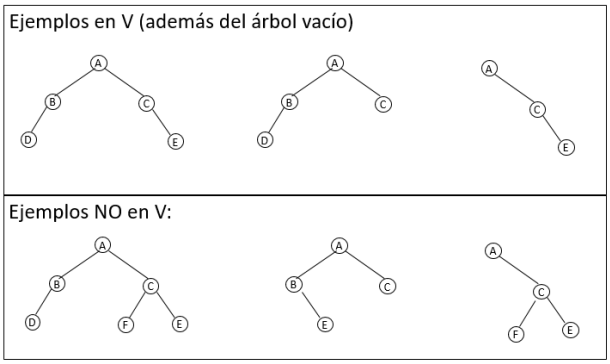
```
Status t_f1(const BTNode *pn, Queue *q, const Element *ele) {
    int cmp;
    Status flag;
    if (pn==NULL) return ERROR;
    flag = q_insert(q, info(pn));
    if (flag == ERROR) return flag;
    cmp = element_cmp (ele, info(pn));
    if (cmp==0) return OK;
    if (cmp < 0)
        return t_f1 (left(pn), pc, ele);   SI
    return t_f1 (right(pn), pc, ele);   SI
}
long int fa(int n){
    if (n <= 1)
        return(1);
    return (n * fa(n-1));   NO
}
void bt_f1 (BTNode *pn) {
    BTNode *temp = NULL;
    if (pn == NULL) return;
    temp = left(pn);
    left(pn) = right(pn);
    right(pn) = temp;
    bt_f1(left(pn));   NO
    bt_f1(right(pn));   SI
}
```

Pregunta 2
Finalizado
Puntúa como 3,00

Proporciona el código en C de una función que determine si un árbol binario tiene forma de V invertida.

Diremos que un árbol binario tiene forma de V invertida cuando o bien es vacío o bien todo nodo de su subárbol izquierdo sólo tiene subárboles izquierdos y todo nodo de su subárbol derecho sólo tiene subárboles derechos.

Observación: la definición anterior no considera la longitud de los lados de la V invertida. Por tanto, un árbol que cumpla con las condiciones indicadas tendrá forma de V, independientemente de la longitud de sus lados.



Instrucciones:

- Usa la plantilla de respuesta proporcionada, rellenando las partes necesarias. **No modifiques** su texto, en otro caso la respuesta puede ser invalidada.
- Si necesitas definir funciones adicionales, escribe **su(s) prototipo(s) y definición(es)** en el espacio proporcionado.
- Escribe delante de cada paso un **comentario** que explique lo que se hace en ese momento con tus palabras. Si no pones comentarios el ejercicio será invalidado.

Puedes usar las siguientes estructuras y macros:

```
/*
 * ----------------------------------------------------------------
 * Estructura para un nodo de un árbol binario.
 * El campo info es un puntero a void para que se puedan almacenar
 * elementos de cualquier tipo.
 * Los campos right y left son punteroa a los nodos hijos del nodo
 * ----------------------------------------------------------------
 */
struct _BTNode {
  void *info;
  struct _BTNode *left;
  struct _BTNode *right;
};

typedef struct _BTNode BTNode;
/*
 * ----------------------------------------------------------------
 * Estructura para un árbol binario.
 * Contiene un puntero al nodo la raiz del árbol.
 * ----------------------------------------------------------------
 */
struct _BSTree {
  BTNode *root;
};

typedef struct _BSTree BSTree;
/*
 * ----------------------------------------------------------------
 * Macros para manejar las partes del árbol binario
 * ----------------------------------------------------------------
 */

#define info(pnode) ((pnode)->info)
#define left(pnode) ((pnode)->left)
#define right(pnode) ((pnode)->right)
#define root(ptree) ((ptree)->root)

/* ----------------------------------------------------------------
 * @brief Función que detecta si un árbol binario tiene forma de V
invertida.
 *
 * @param tree Puntero al árbol binario.
 *
 * @return TRUE si el árbol binario tiene forma de V invertida, o FALSE
 * en caso contrario o si hay algún error
 * ----------------------------------------------------------------
 */


/* Prototipo(s) y definiciones de otra(s) funcion(es) necesarias */


/* comprueba que todo subárbol es vacío o que no tiene hijos derechos: */
Bool is_left_list(BTNode *pn);
```

```
/* comprueba que todo subárbol es vacío o que no tiene hijos izquierdos. */
Bool is_right_list(BTNode *pn);

/* definición de estas funciones */

Bool is_left_list(BTNode *pn){
    /* si el nodo es vacío, hemos llegado al
     * final de la cadena, todo ha ido bien - caso base*/
    if(!pn)
        return TRUE;

    /* si se encuentra un hijo derecho, devolvemos falso,
     * no es un subárbol sin hijos derechos*/
    else if(right(pn) != NULL)
        return FALSE;

    /* caso general de recursión, todo ha ido bien hasta aquí,
     * comprobamos si el subárbol izquierdo es correcto*/
    else
        return is_left_list(left(pn));
}

Bool is_right_list(BTNode *pn){
    /* si el nodo es vacío, hemos llegado al
     * final de la cadena, todo ha ido bien - caso base*/
    if(!pn)
        return TRUE;

    /* si se encuentra un hijo izquierdo, devolvemos falso,
     * no es un subárbol sin hijos izquierdos */
    else if(left(pn) != NULL)
        return FALSE;

    /* caso general de recursión, todo ha ido bien hasta aquí,
     * comprobamos si el subárbol derecho es correcto*/
    else
        return is_right_list(right(pn));
}


/* función principal */
Bool tree_isV (BSTree *tree) {
    /*si el árbol no existe, consideramos que ha habido un error*/
    if(!tree) return FALSE;

    /*si el árbol está vacío, es un árbol en V */
    if(!root(tree)) return TRUE;

    /* devolvemos TRUE si y solo si el subárbol izquierdo es
     * una lista de nodos a la izquierda (sin hijos derechos)
     * y el subárbol derecho una lista de nodos
     * a la derecha (sin hijos izquierdos)*/
    return (is_left_list(left(root(tree))) && is_right_list(right(root(tree))));


}
```

Pregunta 3

Finalizado

Puntúa como 3,00

Implementa el código C de una función que determine si un número *n* está repetido en un array ordenado de números enteros *a*. La función deberá hacer el oportuno control de errores. No se considerarán válidas soluciones no eficientes.

```
/** ---------------------------------------------------------------------
 * @brief Función que detecta si un número está repetido en un array ordenado.
 *
 * @param a Puntero al array.
 * @param l, Tamaño del array
 * @param n, número a detectar
 *
 * @return TRUE si el número está repetido, o FALSE en caso de
 * de que el número no se encuentre en el array o no esté repetido o haya
 * algún error.
 * ---------------------------------------------------------------------- */

Bool estaRep (int *a, size_t l, int n);
```

```
/* Suponemos que el array está ordenado de menor a mayor */

/* función aux: (búsqueda binaria)
 * devuelve la posición del entero n en el array a
 * ó -1 si no lo encuentra o hay algún error.
 * first es la posición del primer elemento del array
 * en el que buscamos en cada momento y last, la última
 */
int busca_numEnArr(int *a, int first, int last, int n);

/*def de función aux*/

int busca_numEnArr(int *a, int first, int last, int n){
    int m;

    if(first > last) return -1;

    m = (first + last) / 2;

    if(a[m] == n) /*si lo encontramos, devolvemos la posición en el array */
        return m;
    else if(a[m] < n) /*entonces buscamos en la mitad superior*/
        return busca_numEnArr(a, m+1, last, n);
    else        /*entonces buscamos en la mitad inferior*/
        return busca_numEnArr(a, first, m-1, n);
}



/*función principal */

Bool estaRep (int *a, size_t l, int n){
    int pos;

    if(!a) return FALSE;
/*buscamos n en a, 0 es la posición inicial y l-1 la última*/
    pos = busca_numEnArr(a, 0, l-1, n);

/*si no se encuentra, devolver FALSE*/
    if(pos == -1) return FALSE;
/*si lo encontramos y es igual a su anterior o posterior, está repetido*/
    else if(a[pos] == a[pos-1] || a[pos] == a[pos+1])
        return TRUE;
/*en caso contrario está pero solamente una vez*/
    else
        return FALSE;
}
```

Pregunta 4

Finalizado

Puntúa como 3,00

[2269658] Completa el código de la función `list_extractFrom` de acuerdo a la documentación de la misma.

Utiliza exclusivamente la información proporcionada sobre las estructuras `Node` y `List`. No supongas la existencia de otras funciones o estructuras.

No hagas ninguna modificación al código proporcionado. Cualquier modificación del código o incluso de los comentarios fuera del cuerpo de la función `list_extractFrom` podrá significar la invalidación del ejercicio.

Es importante que respetes los nombres de las estructuras y campos de las mismas, pues parte de la corrección será automática. En particular tu código será compilado y ejecutado para comprobar su corrección.

```
/**
 * ---------------------------------------------------------------------
 * Estructura para un nodo de una lista.
 * El campo info es un puntero a void para que se puedan almacenar
 * elementos de cualquier tipo.
 * El campo next es un puntero al siguiente nodo de la lista.
 * Al crear una lista será necesario proporcionar punteros a las
 * funciones para liberar, copiar e imprimir un elemento.
 * ---------------------------------------------------------------------
 */
struct _Node {
  void *info;
  struct _Node *next;
};

typedef struct _Node Node;

/**
 * ---------------------------------------------------------------------
 * Estructura para una lista.
 * Contiene un puntero al primer elemento de la lista, first, y
 * punteros a las funciones para liberar, copiar e imprimir un
 * elemento.
 * ---------------------------------------------------------------------
 */
struct _List {
  Node *first;
  P_ele_free pfree;
  P_ele_copy pcopy;
  P_ele_print pprint;
};

typedef struct _List List;

/**
 * ---------------------------------------------------------------------
 * @brief Función que extrae un elemento de una posición arbitraria
 * en una lista.
 *
 * Extrae el elemento en la posición pos de la lista pl, y devuelve
 * el puntero al mismo.
 *
 * @param pl Puntero a la lista.
 * @param pos Entero mayor o igual que 0 que indica la posición en
 * la lista del elemento a extraer.
 *
 * @return Puntero al elemento extraido de la lista, o NULL en caso
 * de error.
 * ---------------------------------------------------------------------
 */
void *list_extractFrom(List *pl, int pos) {
    Node *pn = NULL, *npos = NULL;
    void *ele;
    int i;

    if(!pl || !pl->first) return NULL;

    /* avanzamos en la lista hasta llegar al nodo anterior al que queremos eliminar: */
    for (i=0, pn=pl->first; i<(pos-1) && pn->next != NULL; i++, pn=pn->next);

    if(i != (pos-1)) /*comprobamos si pn es el nodo anterior al que está en la posición pos*/
        return NULL; /*esto significa que no hay un nodo en esta posición porque la lista no es tan larga*/

    /*eliminamos el nodo en pos*/
    npos = pn->next;
    pn->next = npos->next;
    ele = npos->info;
    npos->info = NULL;
    free(npos);

    return ele;
}
```

◄ Parte 1 teoría (TADs, Pilas, Colas)

Control 2 de prácticas - Evaluación Continua (12-13h, 18 Mayo) ►

Volver a: General