Guia de trabajos prácticos N° 5

Árboles binarios

En https://github.com/ayed-unmdp/trees se encuentran ejemplos y templates requeridos en esta guía.

Nomenclatura

- BT - Binary Tree: Árbol binario.
- BTN - binary tree node: nodo de árbol binario.
- SBT - Search Binary Tree: ABB - Árbol binario de búsqueda.
- root: nodo raíz del árbol.
- parent node: nodo padre.
- child node: nodo hijo.
- internal node, inner node, inode, branch node: nodo interno, que tiene hijos.
- external node, leaf node, outer node, terminal node: nodo hoja de un árbol.

```
#define t_elem_btree int

typedef struct _btn {
    t_elem_btree value;
    struct _btn *left;
    struct _btn *right;
} btn;
```

1. Implementar las operaciones básicas para trabajar con árboles binarios
    a. Crear un nodo de un BT en base a un valor t_elem_btree
    b. Eliminar un nodo, si tiene hijos elimina sus hijos también.
    c. Buscar un nodo con un determinado elemento y devolver su referencia. Se busca el nodo en pre-order (debe recorrer todo el árbol cuando no hay un criterio de ordenamiento). Debe devolver la referencia a la ubicación del puntero al nodo. Si no existe, devuelve NULL
    d. Hacer una función que reciba un nodo y devuelva 1 si es hoja y 0 si no lo es.
    e. Contar la cantidad de nodos.
    f. Agrega un nodo en un árbol binario con el siguiente criterio:
        i. Si el subárbol es nulo se agrega ahí,
        ii. Si no es NULL lo agrega en el hijo con menor cantidad de nodos,
        iii. Si sus hijos tienen la misma cantidad de nodos lo agrega a la izquierda.
    g. Agregar un valor con el mismo criterio.
    h. Determinar el nivel de un nodo. Recibe como entrada la raíz, un valor y una función de comparación de valores.
    i. Devolver la altura de un BTN. La altura es la distancia desde las hojas. Cuando el nodo es NULL devuelve -1, la altura de las hojas es 0.
    j. Armar una función para cada tipo de recorrido del árbol (Inorder, postorder, preorder), asumiendo que el elemento del árbol es un puntero a string.
    k. Repetir el ejercicio anterior de modo que reciba una función "***do***" como parámetro de entrada que determine la acción a realizar con el nodo, y un contexto "***ctx***". Luego utilizar esa función para imprimir por consola el contenido del árbol.

2. Utilizando como base la definición anterior, implementar las funciones básicas para trabajar un árbol binario de búsqueda.
    a. Agrega un nodo (***btn****) a un árbol binario de búsqueda (SBT), de modo que no acepte valores repetidos. Utilizar una función ***cmp*** pasada como parámetro que permita comparar el valor de dos ***t_elem_btree***. En el **main** agregar varios valores e imprimir el árbol.
    b. Obtener el puntero que contiene el puntero al nodo con el valor mínimo de un SBT. ¿Es necesario enviar una función de comparación? ¿Por qué?
    c. Obtener el puntero que contiene el puntero al nodo con el valor máximo de un SBT.
    d. Obtener el puntero al nodo con el valor solicitado de un SBT. Debe utilizar una función de comparación pasada por parámetro.
        i. Obtener el puntero al puntero del nodo con el valor solicitado.
        ii. En ambos casos realizar ambas versiones: recursiva e iterativa.
    e. Hacer una función que devuelva 1 si un valor se encuentra en el SBT, o 0 en otro caso.
    f. Quitar un nodo de un SBT, reemplazando el nodo por su rama Derecha y agregando la rama Izquierda a la rama derecha.
    g. Quita un nodo de un SBT, reemplazando el nodo por el máximo de su rama izquierda, o en su defecto por el mínimo de su rama derecha.
    h. Quitar un nodo de un SBT, reemplazando el nodo por:
        i. el máximo de su rama izquierda, si la rama izquierda es igual o más alta que la derecha.
        ii. el mínimo de su rama derecha, si la rama derecha es más alta que la izquierda.
    i. Hacer una función (con cada tipo de reemplazo) para eliminar un valor de un SBT.
        i. ¿Qué método resulta más conveniente? ¿Por qué?
3. Dado un árbol binario con valores enteros, devolver la suma de todos los nodos.
4. Dado un árbol binario con valores enteros, devuelve 1 si todos los nodos tiene un valor igual a la suma de los hijos (exceptuando los nodos hoja), 0 en otro caso.
5. Armar una función que dado un Árbol binario y un valor devuelva el nodo (si está) y además cada vez que un nodo es solicitado intercambia el lugar con su padre (a menos que sea la raíz). El objetivo de esta técnica es que los nodos que se llaman con más frecuencia se encuentren cada vez más accesibles.
6. Armar una función que dado un árbol binario, cree otro exactamente igual pero en espejo.
7. Armar una función (y las funciones auxiliares necesarias) para que dado un Árbol binario (BT) y un SBT (ABB) con los mismos valores sin destruir el árbol original.
8. Eliminar todos los elementos repetidos de una lista dinámica simplemente enlazada, recorriendo la lista una sola vez y utilizando como estructura auxiliar un SBT.
9. Dados 2 SBT, fusionarlos en uno único SBT, eliminado los nodos repetidos.
10. Dado un árbol binario, determinar si es un SBT.
11. Armar 3 funciones que devuelvan el contenido de un árbol en un vector: en inorder, preorder y post order.
12. Armar una función que reconstruya un árbol binario recibiendo como parámetros 2 arreglos con el contenido del árbol en pre-order e in-order.
13. Crear una lista con los valores de los nodos de un árbol binario cuya profundidad es igual a su altura. Resolver el algoritmo recorriendo una sola vez el árbol, y dejando la lista ordenada por nivel de los nodos (es decir los nodos más cercanos a la raíz primero).
14. Convertir, del modo más eficiente, un SBT en una lista dinámica doblemente enlazada y ordenada en forma ascendente. ¿Es posible hacerlo utilizando los mismos nodos del árbol?. ¿Y viceversa?

15. [Entregable] Un sistema distribuído tiene sensores de temperatura en distintos puntos de una ciudad. La estructura de datos de cada medición es la siguiente:

```
typedef struct {
   int minute;
   int temperature;
} reading;
```

En cada punto de medición se encuentra un sensor programado que cada x minutos de cada día (de 0 a 1440) deja las lecturas en un pila (TDA Stack).

Al finalizar el día un proceso crea una lista dinámica enlazada, donde cada elemento es una pila (stack) de lecturas realizadas por un sensor.

Otro proceso requiere recibir como datos de entrada todas las lectura en una queue (TDA Queue), cuyo elemento es la lectura (reading) y deben estar ordenadas por minuto de la medición.

Elaborar una función (y todas las que sean necesarias) que reciba como entrada el puntero a la cabecera de la lista de pilas de lecturas y devuelva como salida una queue con todas las mediciones ordenadas.

Para ello:

- Construir un TDA, básico para las lecturas: crear, borrar, obtener minuto, obtener temperatura y comparar.
- Hacer una función que simule la creación de la pila con distintas lecturas (reading), tomando minutos y temperaturas en forma aleatoria pero coherentes:
  - Los minutos deben ser entre 0 y 1440 y
  - las temperaturas entre -10 y 40 grados, no debería variar más de 5 grados cada lectura.
  - Al menos debe haber entre 20 y 30 lecturas.
- Hacer una función que simule el proceso de ensamblado de varias pilas en una lista dinámica enlazada SLL. Entre 10 y 15 elementos debe tener la lista.
- Hacer la función solicitada separando el problema en 2 partes:
  - Utilizar la librería de árboles binarios de búsqueda (desarrollada en esta guía de trabajos prácticos), cuyo elemento del árbol será la estructura “reading” para insertar todas las lecturas en el árbol ordenando por minutos, en caso de que haya 2 lecturas en el mismo minuto desempatan por temperatura quedando primero la menor, si coinciden en minuto y temperatura se ignora y no se agrega al árbol.
  - Crear la Queue recorriendo el árbol in-orden.
- Para finalizar, hacer una función de testing que lea la queue y verifique que todos los elementos están ordenados.