UNIVERSIDAD DE GRANADA

/ UGR / decsai

Departamento de Ciencias de la Computación e Inteligencia Artificial

Estructuras de datos grupo B

Ejercicios sobre árboles y tablas hash

Esta entrega corresponde a la relación de ejercicios sobre árboles y tablas hash.

- Debajo de cada ejercicio puedes encontrar la función que debes implementar. Además, debe haber una función main con las distintas pruebas que has realizado para verificar el funcionamiento de la función.
- las cabeceras de las funciones deben ser exactamente tal y como aparecen.
- Si un ejercicio no funciona correctamente o se tiene claro que no resuelve todos los casos posibles, se debe explicar adecuadamente.
- La solución debe ser la más eficiente posible. incluida la constante, tanto en tiempo como en espacio.
- Los ficheros no deben dar errores de compilación.
- Los include se deben hacer usando < no “, por ejemplo, #include <bintree.h>

Cualquier entrega que no cumpla con estas normas no será tenida en cuenta.

Ejercicios:

1. Escribe una función que determine cual es la hoja más profunda de un árbol binario.

   typename bintree<T>::node mas_profunda(const bintree<T> &arb)

2. Escribe una función no recursiva (usando una pila) para calcular la altura de un árbol binario.

   int altura(const bintree<T> &arb)

3. Escribe una función a la que se le pasen dos nodos de un árbol y devuelva el primer ancestro común de ambos. La eficiencia debe ser proporcional a la altura del árbol. No se pueden usar iteradores

   typename bintree<T>::node ancestro_comun(typename bintree<T>::node n1, typename bintree<T>::node n2)

4. Escribe una función que realice la reflexión de un árbol binario. Es decir, una función a la que se le pase un árbol y lo modifique de forma que para cada nodo su hijo a la izquierda pase a ser el derecho y viceversa.

   void reflexion(bintree<T> &arb)

5. Dado un árbol de expresión arb cuyos nodos hoja son letras minúsculas del alfabeto y cuyos nodos interiores son los caracteres *, +, -, /. Escribe una función que tome como parámetros un nodo y un árbol binario y devuelva el resultado de la evaluación de la expresión representada considerando los valores asociados a cada letra.

   double evaluar(typename bintree<char>::node n, const bintree<char> &arb, const map<char, double> &valores)

6. El recorrido en preorden de un determinado arbol binario es: GEAIBMCLDFKJH y en inorden IABEGLDCFMKHJ

   Suponemos que todas las letras son diferentes.

   Realiza las siguientes tareas:

   1. Dibuja el árbol binario (inclúyelo entre comentarios).
   2. Da el recorrido en postorden (inclúyelo entre comentarios).
   3. Diseña una función para dar el recorrido en postorden dados los recorridos en preorden e inorden (sin construir el árbol) y escribe un programa para comprobar el resultado del apartado anterior.

      string postorden(string preorden, string inorden)

7. Construye una función que determine si un árbol binario es completo. Es decir que tenga todos los niveles completos.

   bool completo(const bintree<T> &arb)

8. Dada una expresión en postfijo generar el árbol asociado, de forma que el recorrido en postorden del mismo devuelva la expresión original. La cadena solo tiene caracteres y operadores, sin separadores.

   bintree<T> construye_arbol(string postfijo)

9. Dos árboles binarios los definimos similares si son iguales en cuanto a su estructura, es decir, cada nodo en un árbol tiene los mismos hijos y en el mismo lugar que el correspondiente en el otro árbol (sin importar las etiquetas). Escribe una función que dados dos árboles binarios devuelva si son o no similares.

   bool similares(const bintree<T> &arb1, const bintree<T> &arb2)

10. Construye un BST y un POT con las claves 50,25,75,10,40,60,90,35,45,70,42. Indicando todos los pasos de forma detallada.
11. Indica la secuencia de rotaciones resultante de la inserción del conjunto de elementos {1, 2, 3, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9, 8} en un árbol AVL.
12. ¿Bajo qué condiciones puede un árbol ser parcialmente ordenado y binario de búsqueda simultáneamente? Razona la respuesta.
13. Se define la densidad de un árbol binario como la suma de las profundidades de sus hojas dividido por la altura del árbol. Construye una función para calcular la densidad de un árbol binario.

    double densidad(const bintree<T> &arb)

14. Dado un bintree, con las etiquetas organizadas como un BST, implementa una función a la que se le pasen dos valores a y b y que determine de forma eficiente los valores presentes en el árbol y que estén comprendidos entre ambos (ambos inclusive). Tanto a como b no tienen porqué aparecer en el árbol. No se pueden usar iteradores ni hacer ningún recorrido.

    list<T> intervalo(const bintree<T> &arb, const T &a, const T &b)

15. Dada una secuencia de claves S = {5, 8, 4, 13, 66, 2, 9, 12, 11, 17}, inserta la secuencia anterior, en el orden indicado, en una tabla hash cerrada con resolución de colisiones lineal y que tiene tamaño 11. A continuación, borra los elementos 2 y 17. Construye el árbol AVL que se obtiene al insertar los elementos de la secuencia S (en el orden en que aparecen).

16. Implementa una clase tabla_hash<T> que use como representación vector<set<T> > .
Implementa la clase iterator con las operaciones operator--(int),operator++(int), ==, !=, *

```
class tabla_hash {
  public:
    void insert(const T &x);
    void erase(const T &x);
    bool find(const T &x) const;
    iterator begin() const;
    iterator end() const;
…
```

Se debe usar hash<T> . El tamaño de la tabla debe empezar con tamaño M=13 y cambiar a 2*M+1 cada vez que se llene al 50%