Estructuras de Datos 2020/21 – Segundo Parcial
18 de enero de 2021
15:00 – 17:00
Aulas: P2.01, P2.04, P2.09 y P2.11

Pregunta 1. (2,5 puntos)

Dada una tabla hash vacía, como la de la Figura 1(a), dibuja paso a paso, cómo queda con las operaciones indicadas en la Figura 1 (b). Es necesario explicar brevemente qué ocurre y cómo se actúa en cada nueva situación, es decir, la primera vez que ocurre una nueva situación, deberá ser explicada pero no en las siguientes veces que ocurra.

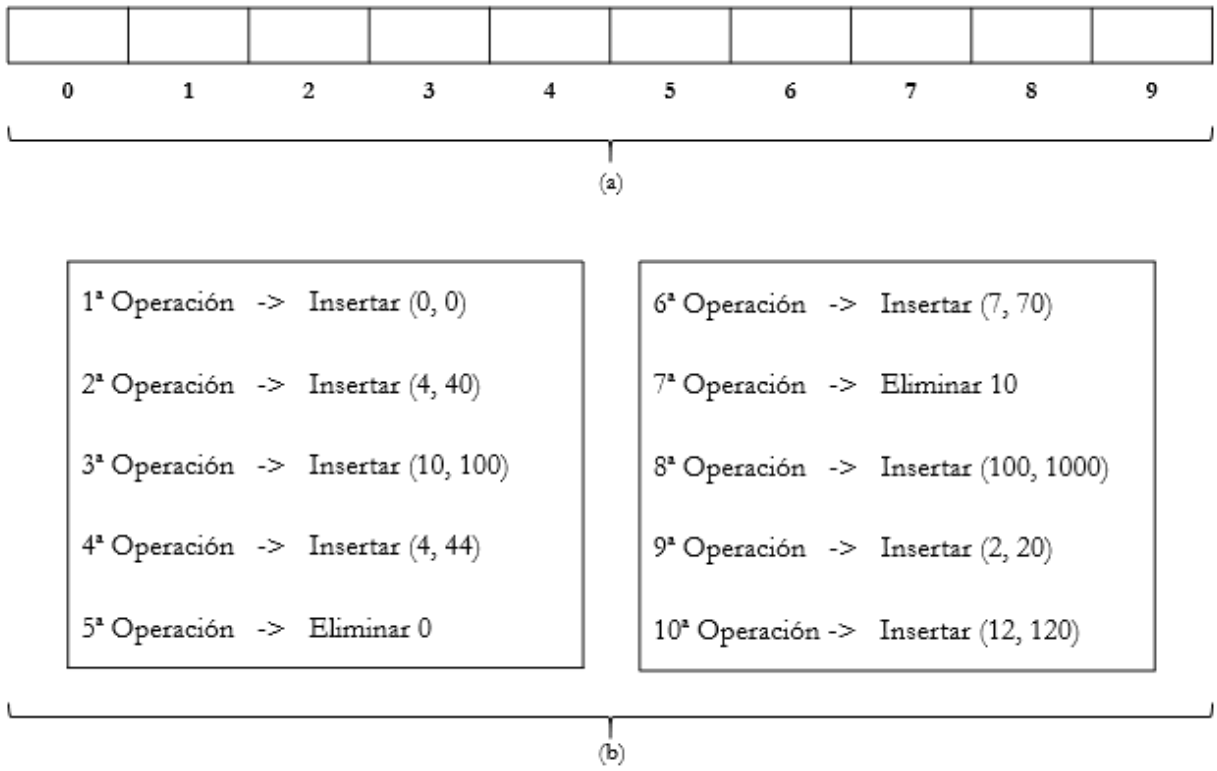

Figura 1.Tabla (a) y operaciones (b) de la pregunta 1

Información sobre la tabla:

- Función de dispersión a utilizar:
  *h(k) = (3 * k + 1) % (tamaño de la tabla)*
  Ejemplo (para una tabla de tamaño 10)
  *h(30) = (3 * 30 + 1) % 10 = 91 % 10 = 1*
- Estrategia de dispersión utilizada: dispersión abierta.
- La tabla de dispersión se representa por medio de un vector de nodos enlazados de parejas (clave, valor).

<u>Pregunta 2</u>. (2,5 puntos)

Considerando el árbol B de la Figura 2, realiza las siguientes operaciones:

- Eliminar la clave 14 sobre el árbol B original.
- Añadir la clave 90 sobre el árbol B original.

<u>Explica</u> las situaciones que se dan en cada momento. Debes explicar el proceso seguido, así como los pasos y resultados intermedios para llegar al resultado final.

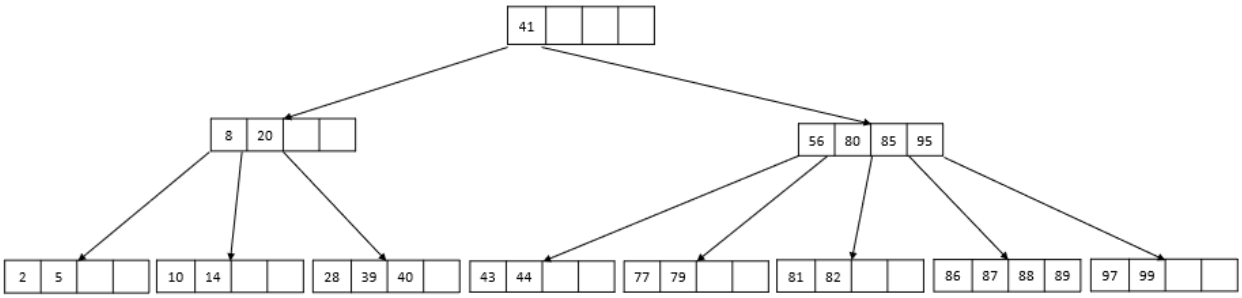

Figura 2. Árbol *B* de la pregunta 2

<u>Pregunta 3</u>. (2,5 puntos)

Implementa un método, llamado *sumAndSubtract*, que reciba un *BinaryTree<Integer>* (llamado *original*), un entero llamado *sum* y otro entero llamado *sub*. Este método debe devolver un *LinkedBinaryTree<Integer>* en el que cada posición del nuevo árbol contenga el resultado de sumar *sum* al nodo equivalente del árbol inicial si su valor es par o de restarle *sub* si su valor es impar. En el caso de una hoja en el árbol original, su equivalente en el nuevo árbol será el resultado de restarle a la hoja el valor *sub* y sumarle *sum,* independientemente de si la hoja es par o impar.

Si el árbol recibido como parámetro es vacío, la operación devolverá un árbol vacío.

Con el fin de tener más clara la operación a realizar, la Figura 3 contiene un ejemplo de uso del método *sumAndSubtract*.

La implementación del método debe ser realizada **<u>fuera</u>** de la clase *LinkedBinaryTree<E>*.

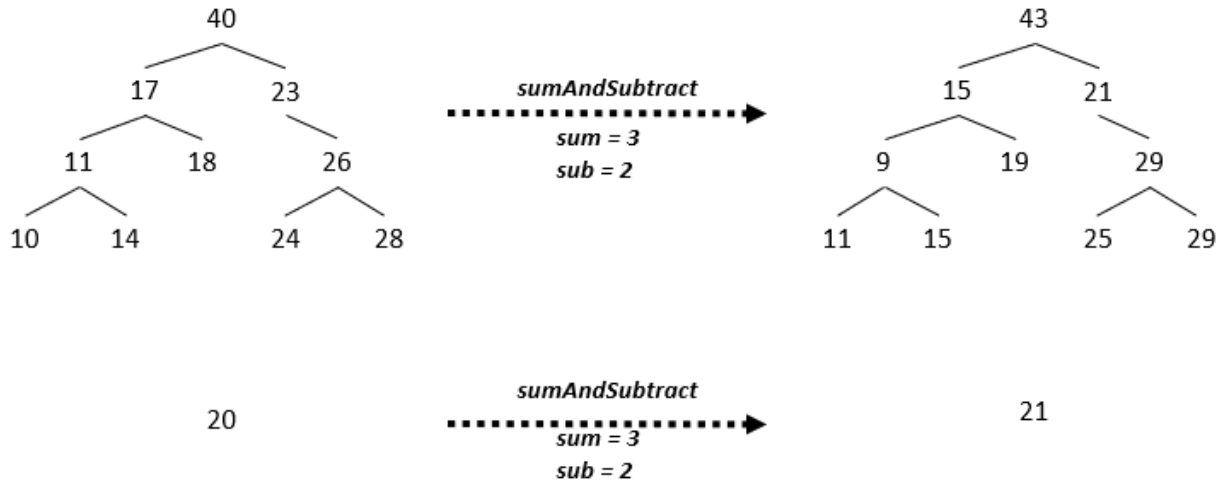

Figura 3. Ejemplo relativo a la operación *sumAndSubtract* (Pregunta 3)

Pregunta 4. (2,5 puntos)

Dado el árbol binario de búsqueda inmutable de la Figura 4, explicar paso a paso, la inserción (*put*) de la pareja clave-valor (15, 150) y la eliminación (*remove*) de la clave 12. No se pide realizar la implementación, se pide explicar textual y gráficamente (diagramas) el proceso, siendo ambas explicaciones necesarias. Muestra claramente los nodos compartidos (y los que no) con el árbol original.

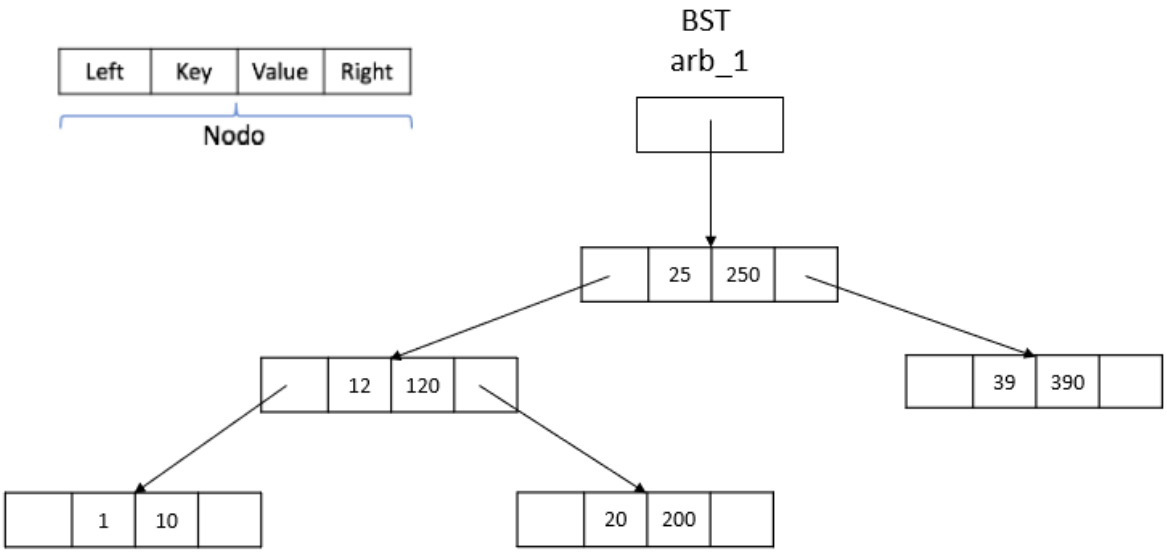

Figura 4. Árbol binario de búsqueda inmutable correspondiente a la pregunta 4

Métodos de utilidad

```
public interface BinaryTree<E> extends Collection<E> {
    BinaryTree<E> getLeftCh();
    BinaryTree<E> getRightCh();
    E getRoot();
    boolean isEmpty();
    //. . .
}

public class LinkedBinaryTree<E>
        extends AbstractCollection<E> implements BinaryTree<E>{
    //. . .
    LinkedBinaryTree(){ . . . }
    LinkedBinaryTree(LinkedBinaryTree<E> left,
                     E elem,
                     LinkedBinaryTree<E> right) { . . . }

    //. . .
}
```