Práctica 02

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Maestría en Ciencia de la Computación	Algoritmos y Estructura de Datos

PRÁCTICA	TEMA	DURACIÓN
02	Estructuras de datos	3 horas

1. Datos de los estudiantes

- Grupo: V
- Integrantes:
  - Angel Yvan Choquehuanca Peraltilla
  - Estefany Pilar Huaman Colque
  - Eduardo Diaz Huayhuas
  - Gustavo Raul Manrique Fernandez

2. Introducción

En el caso de árboles binarios, si los árboles están sesgados, se vuelven computacionalmente ineficientes para realizar operaciones en ellos. Esta es la motivación detrás de asegurarse de que los árboles no estén sesgados. De ahí la necesidad de árboles binarios balanceados.

3. Marco Teorico

Árbol de búsqueda binario autoequilibrado

3.1. Estructura de datos AVL

Un árbol AVL es un árbol de búsqueda binario balanceado. En un árbol AVL, el factor de equilibrio de cada nodo es -1, 0 o +1.

El factor de equilibrio de un nodo es la diferencia entre las alturas de los subárboles izquierdo y derecho de ese nodo. El factor de equilibrio de un nodo se calcula ya sea altura del subárbol izquierdo - altura del subárbol derecho (O) altura del subárbol derecho - altura del subárbol izquierdo . En la siguiente explicación, calculamos de la siguiente manera:

Factor de equilibrio = altura del subárbol izquierdo - altura del subárbol derecho

En el árbol AVL, después de realizar operaciones como inserción y eliminación, debemos verificar el factor de equilibrio de cada nodo en el árbol. Si todos los nodos satisfacen la condición del factor de equilibrio, concluimos la operación; de lo contrario, debemos equilibrarlo. Cada vez que el árbol se desequilibra debido a alguna operación, utilizamos operaciones de rotación para equilibrar el árbol.La rotación es el proceso de mover los nodos hacia la izquierda o hacia la derecha para equilibrar el árbol.

Hay cuatro rotaciones y se clasifican en dos tipos:

- Rotación izquierda
- Rotación a la derecha
- Rotación izquierda-derecha
- Rotación derecha-izquierda

1. **Rotación Simple a la izquierda (rotación LL)** En Rotación LL, cada nodo se mueve una posición a la izquierda desde la posición actual. Para comprender la Rotación LL, consideremos la siguiente operación de inserción en el Árbol AVL.

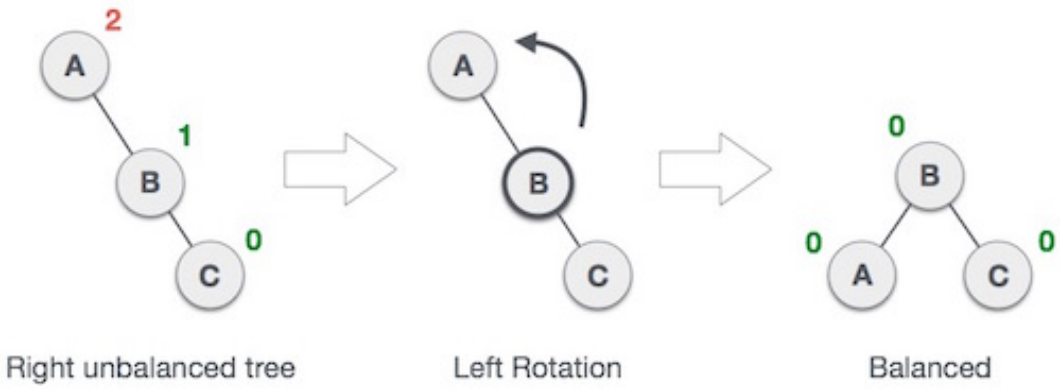

Figura 1: Rotación LL

2. **Rotación Simple a la derecha (rotación RR)** n Rotación RR, cada nodo se mueve una posición a la derecha desde la posición actual. Para comprender la rotación RR, consideremos la siguiente operación de inserción en el árbol AVL.

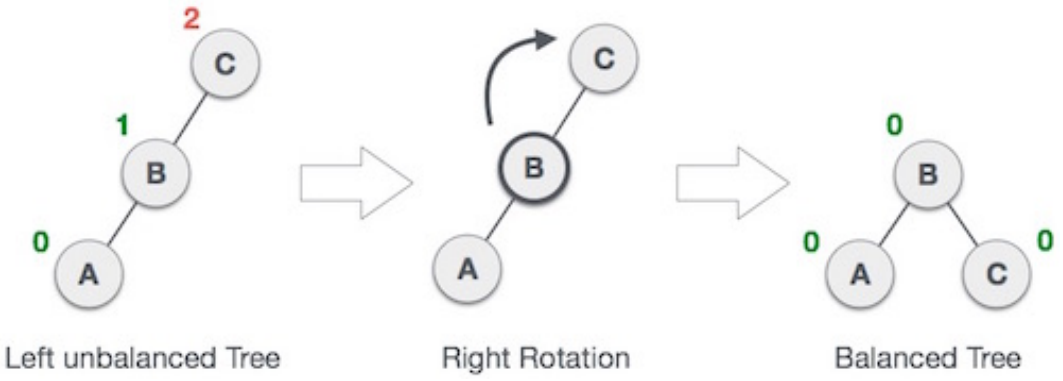

Figura 2: Rotación RR

3. **Rotación izquierda derecha (rotación LR)** La rotación LR es una secuencia de una sola rotación a la izquierda seguida de una sola rotación a la derecha. En LR Rotation, al principio, cada nodo se mueve una posición a la izquierda y una posición a la derecha desde la posición actual. Para comprender la Rotación LR, consideremos la siguiente operación de inserción en el Árbol AVL.

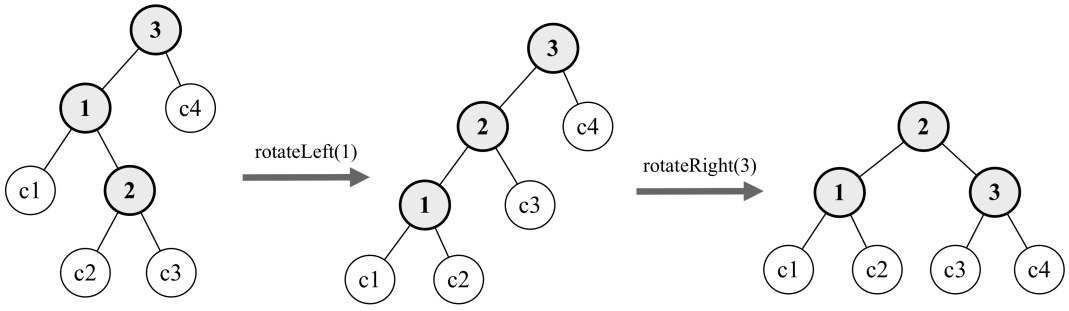

Figura 3: Rotación LR

4. **Rotación derecha izquierda (rotación RL)** La rotación RL es una secuencia de una sola rotación a la derecha seguida de una sola rotación a la izquierda. En Rotación RL, al principio cada nodo se mueve una posición a la derecha y una posición a la izquierda desde la posición actual. Para comprender la Rotación RL, consideremos la siguiente operación de inserción en el Árbol AVL.

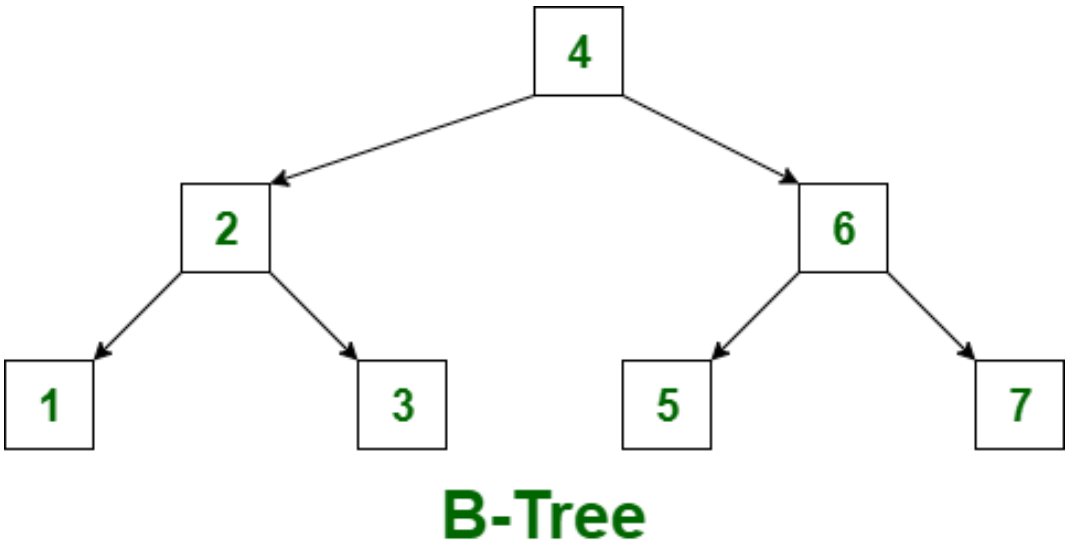

Figura 4: Rotación RL

Las siguientes operaciones se realizan en el árbol AVL:

1. **Operación Búsqueda** En un árbol AVL, la operación de búsqueda se realiza con una complejidad de tiempo O(log n) . La operación de búsqueda en el árbol AVL es similar a la operación de búsqueda en un árbol de búsqueda binario. Usamos los siguientes pasos para buscar un elemento en el árbol AVL:

   - Paso 1: lee el elemento de búsqueda del usuario.
   - Paso 2: compare el elemento de búsqueda con el valor del nodo raíz en el árbol.
   - Paso 3: Si ambos no coinciden, verifica si el elemento de búsqueda es más pequeño o más grande que el valor de ese nodo.

- Paso 4: si el elemento de búsqueda es más pequeño, continúa el proceso de búsqueda en el subárbol izquierdo.
- Paso 5: si el elemento de búsqueda es más grande, continúa el proceso de búsqueda en el subárbol derecho.
- Paso 6: Repite lo mismo hasta encontrar el elemento exacto o hasta comparar el elemento de búsqueda con el nodo hoja.
- Paso 7: si llega al nodo que tiene el valor igual al valor de búsqueda y finaliza la función.
- Paso 8: si llega al nodo de hoja y tampoco coincide con el elemento de búsqueda, finaliza la función.

2. **Operación Inserción** En un árbol AVL, la operación de inserción se realiza con una complejidad de tiempo O(log n) . En AVL Tree, siempre se inserta un nuevo nodo como un nodo hoja. La operación de inserción se realiza de la siguiente manera:
   - Paso 1: inserte el nuevo elemento en el árbol utilizando la lógica de inserción del árbol de búsqueda binaria.
   - Paso 2: después de la inserción,se verifica el factor de equilibrio de cada nodo.
   - Paso 3: si el factor de equilibrio de cada nodo es 0, 1 o -1 , va a la siguiente operación.
   - Paso 4: si el factor de equilibrio de cualquier nodo es distinto de 0, 1 o -1 , se dice que ese árbol está desequilibrado. En este caso, realiza la Rotación adecuada para equilibrarlo y va a la siguiente operación.
3. **Operación Eliminación** La operación de eliminación en AVL Tree es similar a la operación de eliminación en BST. Pero después de cada operación de eliminación, debemos verificar la condición del factor de equilibrio. Si el árbol está equilibrado después de la eliminación, vaya a la siguiente operación; de lo contrario, realice una rotación adecuada para equilibrar el árbol.

3.2. Estructura de datos B Tree

B-Tree es un árbol de búsqueda autoequilibrado en el que cada nodo contiene varias claves y tiene más de dos elementos secundarios. Donde el número de claves en un nodo y el número de hijos para un nodo depende del orden de B-Tree. Cada B-Tree tiene un orden.

B-Tree de Orden m tiene las siguientes propiedades:

- Propiedad n.º 1 : todos los nodos de hoja deben estar al mismo nivel .
- Propiedad n.º 2 : todos los nodos, excepto el raíz, deben tener al menos [m/2]-1 claves y un máximo de m-1 claves.
- Propiedad n.º 3 : todos los nodos que no son hojas excepto la raíz (es decir, todos los nodos internos) deben tener al menos m/2 hijos.
- Propiedad n.º 4 : si el nodo raíz no es un nodo hoja, debe tener al menos 2 hijos.
- Propiedad n.º 5 : un nodo que no sea hoja con n-1 claves debe tener n número de hijos.
- Propiedad n.º 6 : todos los valores clave de un nodo deben estar en orden ascendente .

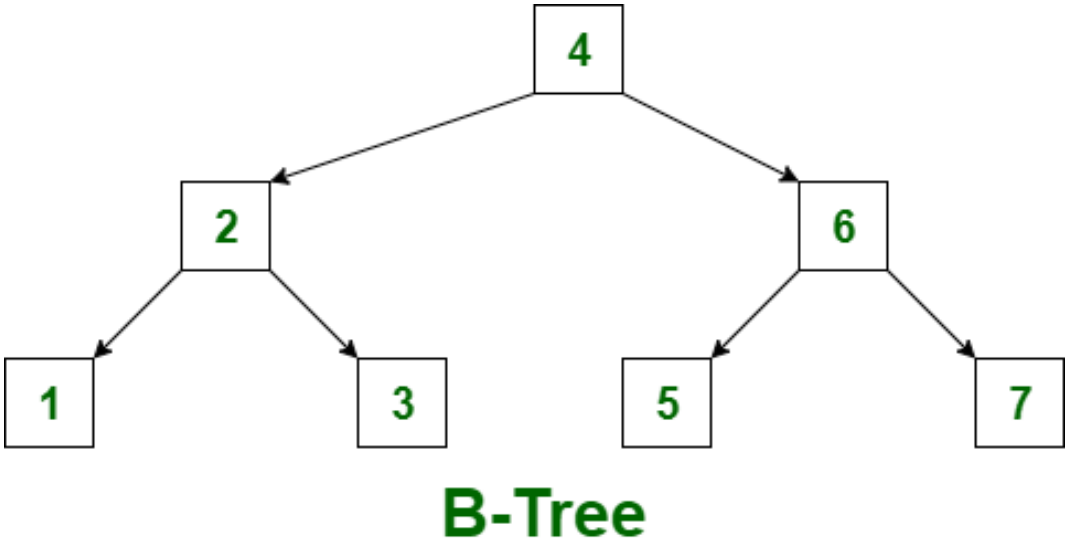

Figura 5: Ejemplo de Estructura de datos B Tree

Las siguientes operaciones se realizan en un B-Tree:

1. **Operación Búsqueda** La operación de búsqueda en B-Tree es similar a la operación de búsqueda en Binary Search Tree. En un árbol de búsqueda binario, el proceso de búsqueda comienza desde el nodo raíz y tomamos una decisión bidireccional cada vez (vamos al subárbol izquierdo o al subárbol derecho). En B-Tree también el proceso de búsqueda comienza desde el nodo raíz, pero aquí siempre tomamos una decisión de n vías. Donde 'n' es el número total de hijos que tiene el nodo. En un B-Tree, la operación de búsqueda se realiza con una complejidad de tiempo O(log n) . La operación de búsqueda se realiza de la siguiente manera:

   - Paso 1: lee el elemento de búsqueda del usuario.
   - Paso 2: compara el elemento de búsqueda con el primer valor clave del nodo raíz en el árbol.
   - Paso 3: si ambos coinciden, se muestra el nodo encontrado y termina la función.
   - Paso 4: si ninguno de los dos coincide, comprueba si el elemento de búsqueda es más pequeño o más grande que el valor de la clave.
   - Paso 5: si el elemento de búsqueda es más pequeño, continúa el proceso de búsqueda en el subárbol izquierdo.
   - Paso 6: si el elemento de búsqueda es más grande, compara el elemento de búsqueda con el siguiente valor clave en el mismo nodo y repite los pasos 3, 4, 5 y 6 hasta que encontrar la coincidencia exacta o hasta que el elemento de búsqueda se compare con el último valor clave en el nodo hoja.
   - Paso 7: si el último valor clave en el nodo hoja tampoco coincide, finalice la función.

2. **Operación Inserción** En un B-Tree, se debe agregar un nuevo elemento solo en el nodo hoja. Eso significa que el nuevo valor clave siempre se adjunta solo al nodo hoja. La operación de inserción se realiza de la siguiente manera:

   - Paso 1: comprueba si el árbol está vacío.
   - Paso 2: si el árbol está vacío , crea un nuevo nodo con un nuevo valor de clave y lo inserta en el árbol como un nodo raíz.

- Paso 3: si el árbol no está vacío , busca el nodo de hoja adecuado al que se agrega el nuevo valor clave utilizando la lógica del árbol de búsqueda binaria.
- Paso 4: si ese nodo hoja tiene una posición vacía, agrega el nuevo valor clave a ese nodo hoja en orden ascendente de valor clave dentro del nodo.
- Paso 5: si ese nodo de hoja ya está lleno, divide ese nodo de hoja enviando el valor medio a su nodo principal. Repite lo mismo hasta que el valor de envío se fije en un nodo.
- Paso 6: si el derrame se realiza en el nodo raíz, el valor medio se convierte en un nuevo nodo raíz para el árbol y la altura del árbol aumenta en uno.

4. Metodologia y Desarrollo

4.1. Codigo Fuente

Para el desarrollo del codigo se utilizó el lenguaje JavaScript, donde se utiliza la libreria p5.js

4.2. Despliegue en la nube

Para la realización de este proyecto, se utilizó los servicios en la nube de Heroku www.heroku.com, bajo la direcciones:

- B-Tree: Heroku B-Tree
- AVL-Tree: Heroku AVL-Tree

El repositorio donde se encuentran los archivos fuente se encuentra en: Github - MCC - Proyecto 2

4.3. Ejecución del Programa

4.3.1. B-Tree

Para el B-Tree se tiene un panel en la izquierda donde se puede hacer las operaciones siguientes:

- Insert (Insertar)
- Search (Buscar)
- Delete (Borrar)

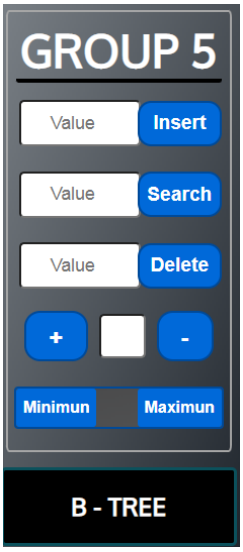

Figura 6: Panel de Control de B-Tree

Se deben colocar los datos en las cajas respectivas y hacer clic en la operación que se desea realizar. Luego se visualiza la formación del arbol en la parte derecha de la aplicación.

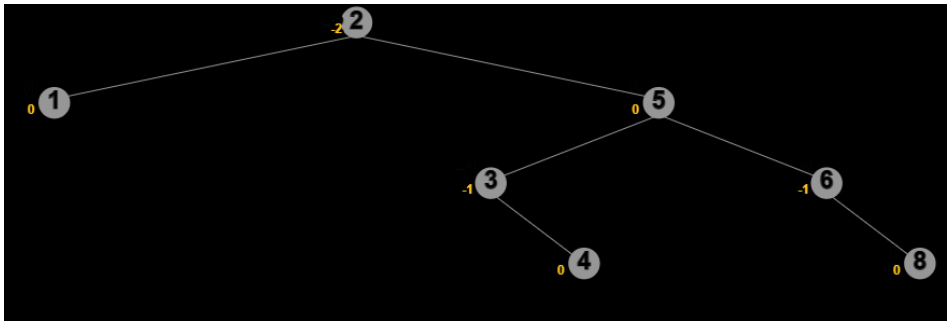

Figura 7: Formacion del arbol B-Tree

Si hay una duplicación el programa en el Status Bar mostrará un error:

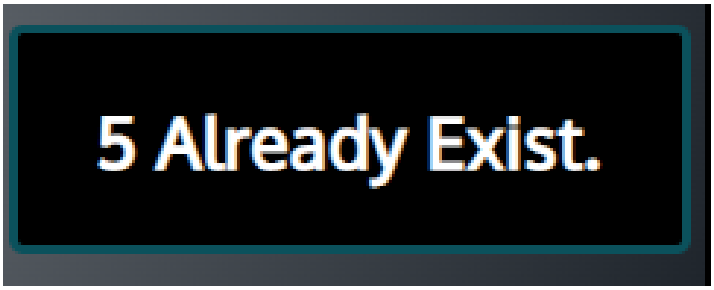

Figura 8: Error de Existencia

En el caso que no exista el dato, se procede a insertar y se muestra el mensaje siguiente:

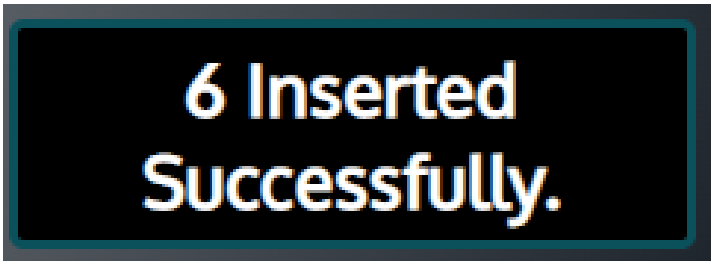

Figura 9: Mensaje de Afirmacion

4.3.2. AVL-Tree

Para AVL-Tree tambien se tiene un panel en la izquierda donde se puede hacer las operaciones siguientes:

- Insert (Insertar)
- Search (Buscar)
- Delete (Borrar)

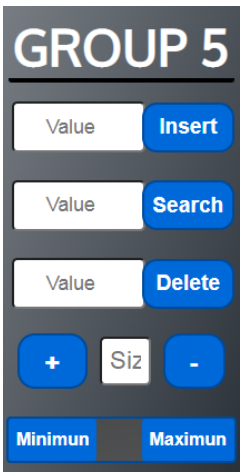

Figura 10: Panel de Control de AVL-Tree

Las mismas operaciones que se encuentran en B-Tree tambien se realizan en AVL-Tree.
Si en caso se desea eliminar un dato, se mostrará el siguiente mensaje:

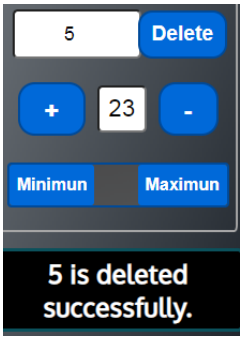

Figura 11: Eliminacion en AVL-Tree

Y si se usa el boton SEARCH y se encuentra el dato, mostrará el mensaje de "found".

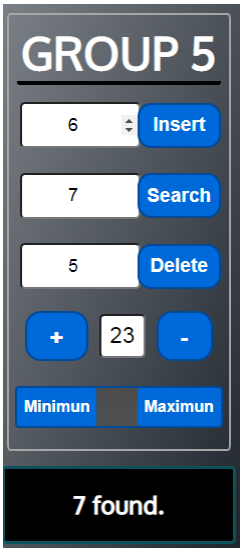

Figura 12: Busqueda en AVL-Tree

5. Resultados

5.1. B-Tree

Para B-Tree se puede desarrollar arboles de una manera facil utilizando la libreria P5.js . Los botones + y - permiten aumentar y disminuir el tamaño del arbol para una mejor visualización.

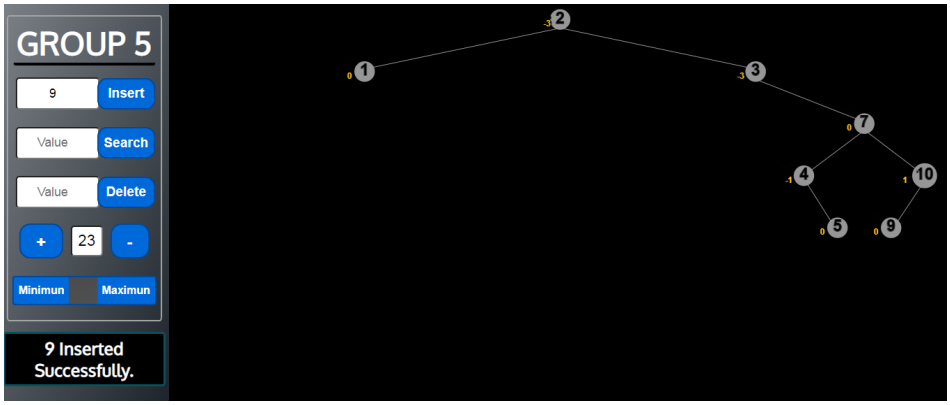

Figura 13: B-Tree en Heroku

5.2. AVL-Tree

Para AVL-Treee también se puede desarrollar arboles de una manera facil utilizando la libreria P5.js . Los botones + y - permiten aumentar y disminuir el tamaño del arbol para una mejor visualización.

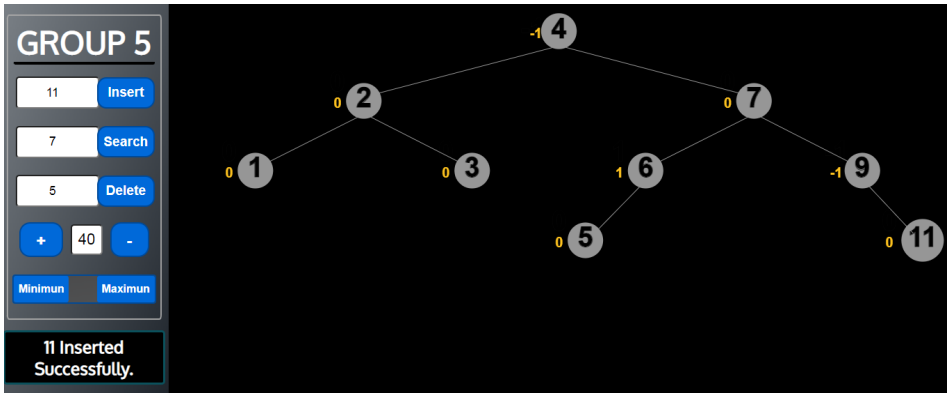

Figura 14: AVL-Tree en Heroku

6. Conclusiones

6.1. B-Tree y AVL-Tree

- Logramos identificar que, entre las operaciones disponibles (Insert, Serach, Delete, Maximun y Minimun). Para AVL-Tree solo las operaciones de Insert y Delete necesitan ser balanceadas.
- Las aplicaciones nos permiten diferenciar que para B-Tree no se balancea, mientras que un AVL-Tree sí se balancea.
- Un AVL es más eficiente porque el balanceo nos permite asegurarnos que la complejidad simepre será nlog(n). Mientras que en el B-Tree se puede presentar el caso en que la complejidad llegue a "n"por no balancearse.