Métodos algorítmicos en resolución de problemas I

Implementación de un *Splay Tree*

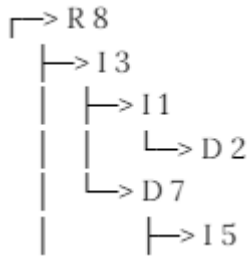
1. Implementación

La implementación de un árbol de búsqueda autoajustable se ha realizado según los apuntes de la asignatura. El origen de flotación, es el que se especifica en los apuntes de árboles autoajustables.

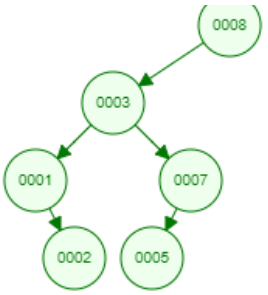
2. Casos de prueba

2.1. Inserción en un árbol vacío de las claves: 2, 7, 3, 5, 1, 3, 8 en ese orden

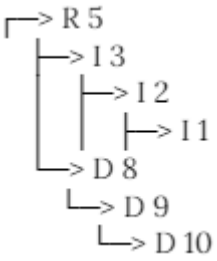
Resultado según la implementación[1]

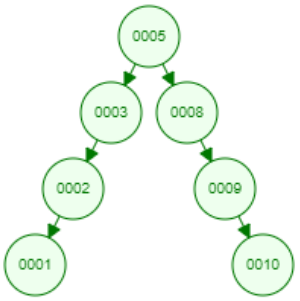


Resultado esperado[2]



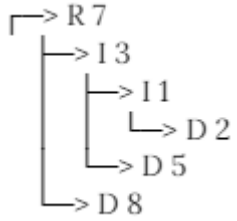
2.2. Inserción en un árbol vacío de las claves: 10, 1, 9, 2, 8, 3, 5 en ese orden



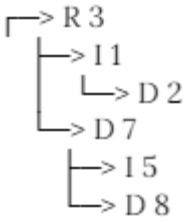


2.3. Búsqueda de los valores 7,3 y 2 en el árbol del apartado 2.1 (en ese orden)

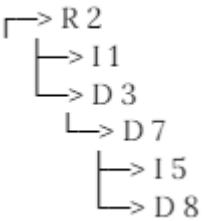
Búsqueda del 7



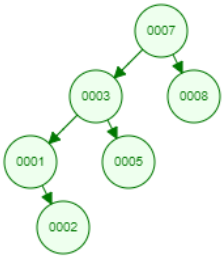
Búsqueda del 3

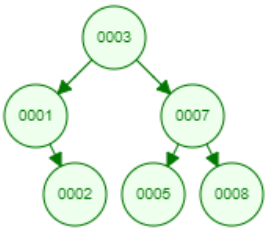


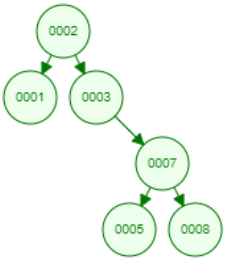
Búsqueda del 2



[1] 'R' para raíz, 'I' para el elemento a su izquierda y 'D' para el elemento a su derecha
[2] Obtenido con: https://www.cs.usfca.edu/~galles/visualization/SplayTree.html

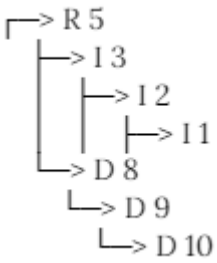


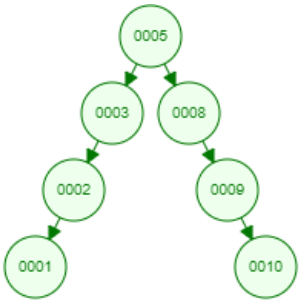




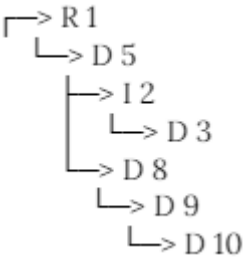
2.4. Búsqueda de los valores 5,1 y 9 en el árbol del apartado 2.2 (en ese orden)

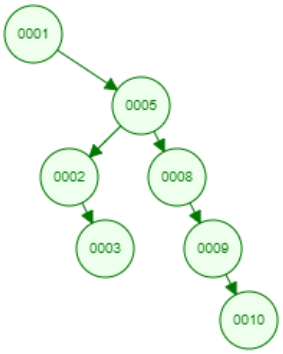
Búsqueda del 5



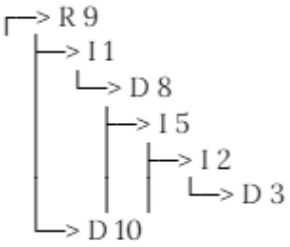


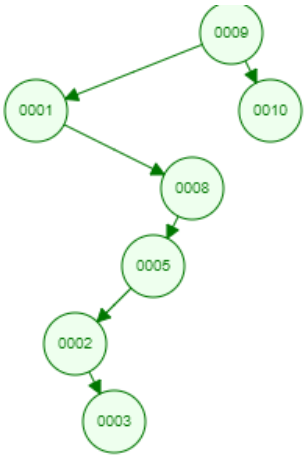
Búsqueda del 1





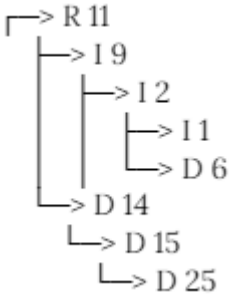
Búsqueda del 9

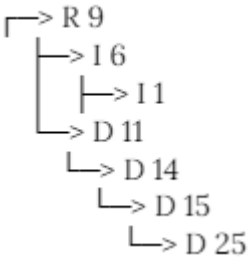


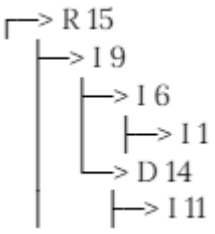


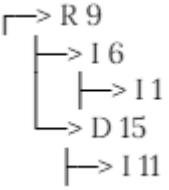
2.5. Borrado[3]

Árbol inicial[4] -> Borrar 2 (nodo intermedio) -> Borrar 25 (hoja) -> Borrar 14 (1 hijo)





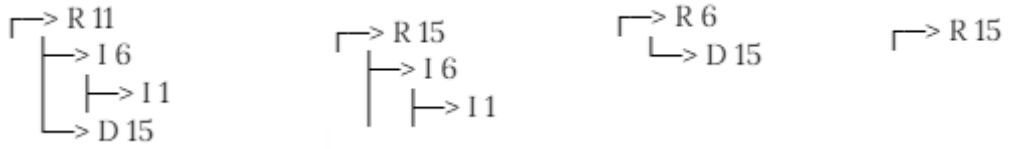




[3] Para el borrado no se ha comparado la solución como en el insertar y en el buscar debido a que en esta implementación se flota el padre del nodo borrado, a diferencia de la implementación de usfca.

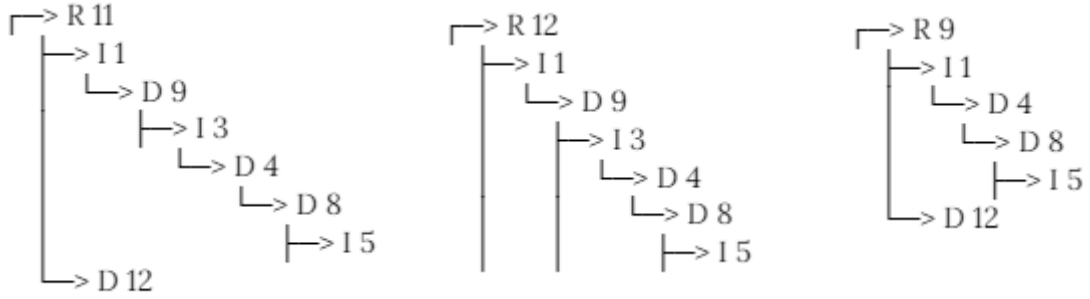
[4] Orden de inserción: 1, 14, 25, 6, 15, 2, 9, 11

-> Borrar 9 (raíz) -> Borrar 11 (raíz) -> Borrar 1 (hoja) -> Borrar 6 (raíz)



2.6. Borrado

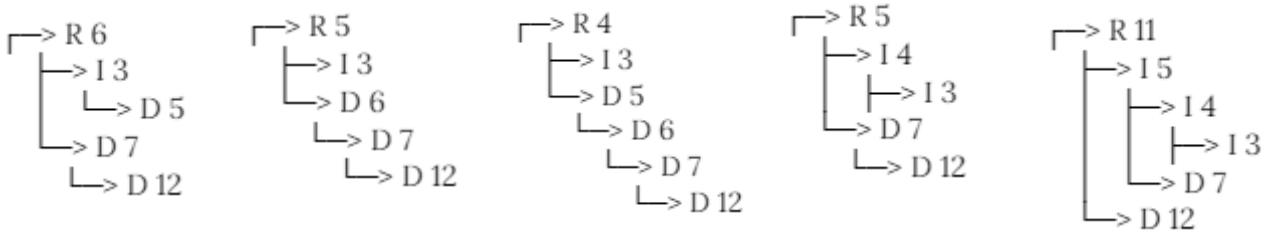
Árbol inicial[5] -> Borrar 11 -> Borrar 3



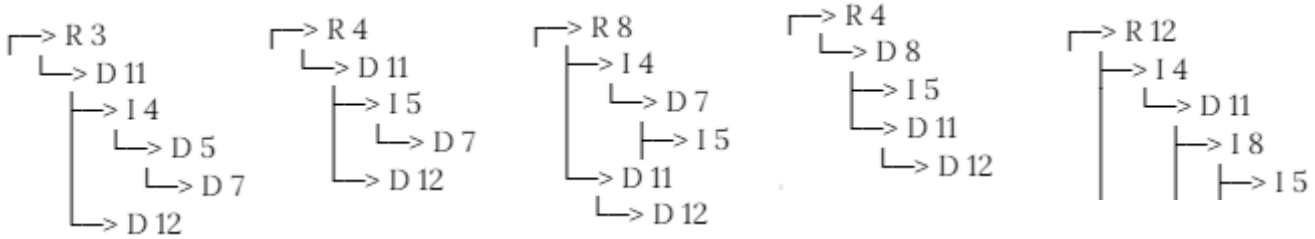
-> Borrar 1 -> Borrar 9

2.7. Inserción, borrado y búsqueda

Árbol inicial[6] -> Buscar 5 -> Insertar 4 -> Borrar 6 -> Insertar 11

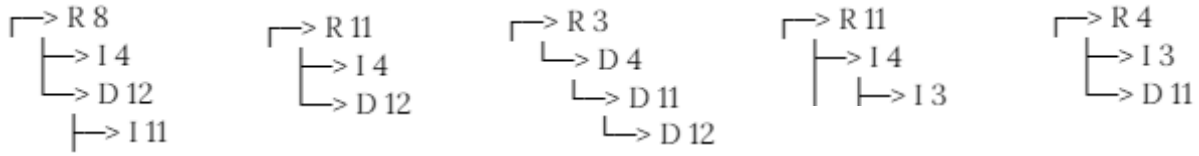


-> Buscar 3 -> Borrar 3 -> Insertar 8 -> Borrar 7 -> Buscar 12



-> Borrar 5 -> Borrar 8 -> Insertar 3 -> Borrar 12 -> Buscar 4

[5] Orden de inserción: 8, 3, 5, 4, 9, 12, 1
[6] Orden de inserción:12, 5, 7, 3, 6



3. Gráficas de tiempo

3.1. Tiempos de n operaciones

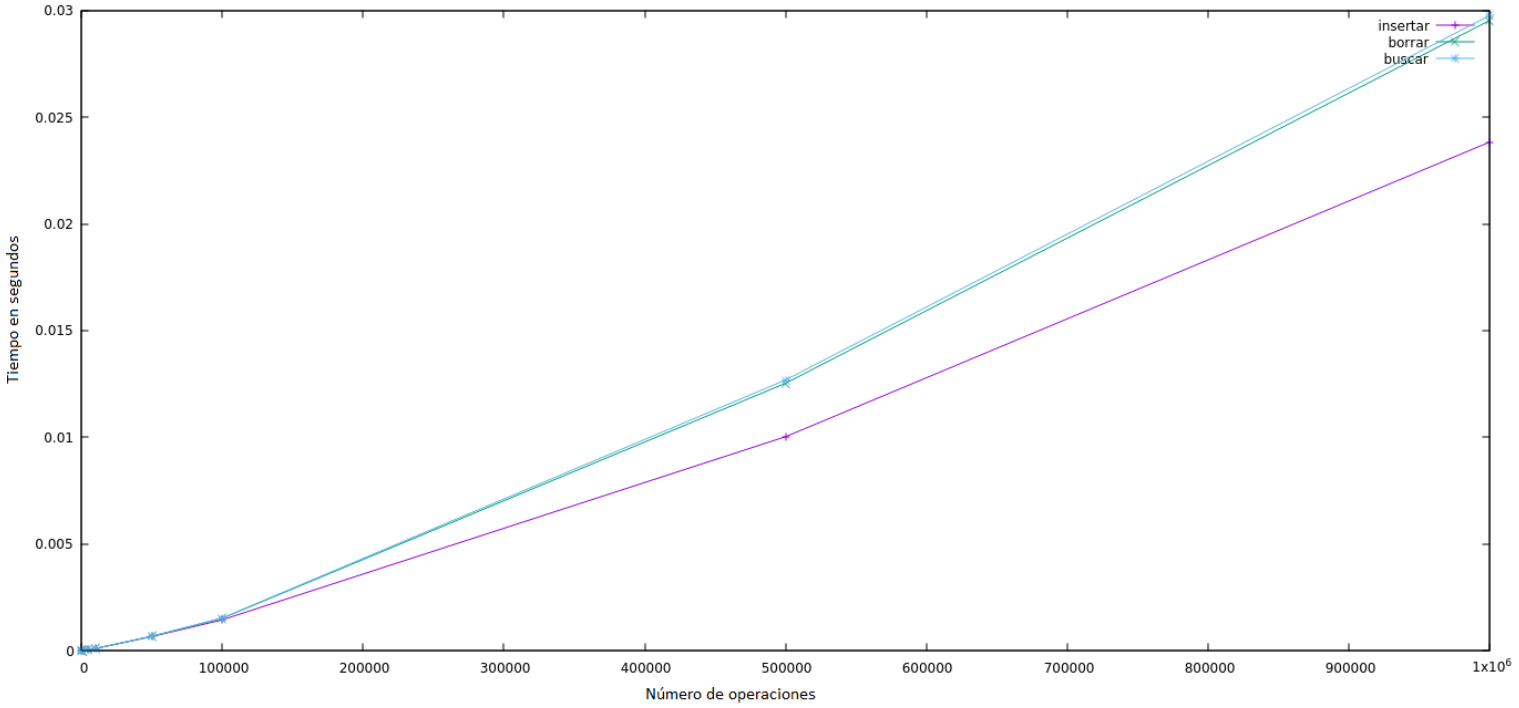


Gráfico 1: se puede encontrar en los archivos del programa como "Insertar_borrar_buscar_n_operaciones.png"

3.2. Tiempos de cada operación

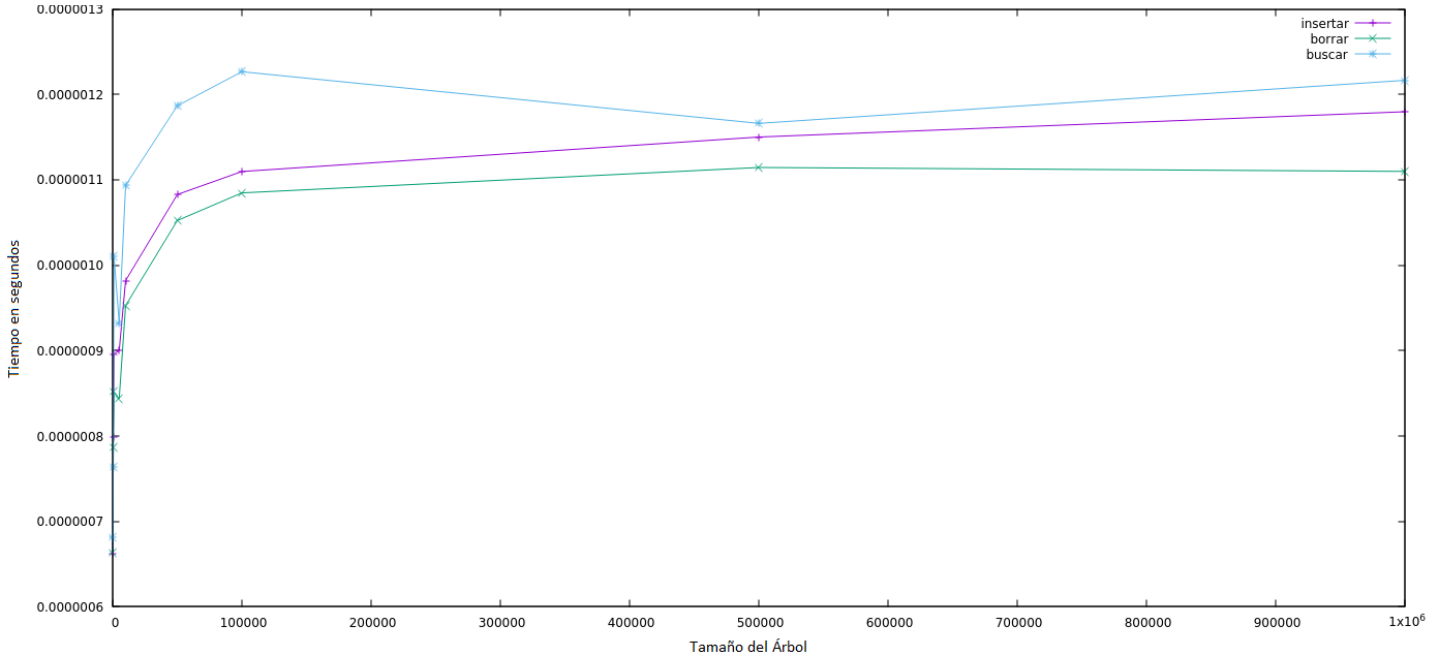


Gráfico 2: se puede encontrar en los archivos del programa como "insertar_borrar_buscar_arbol_tam_n.png"

3.3. Tiempos de las tres operaciones

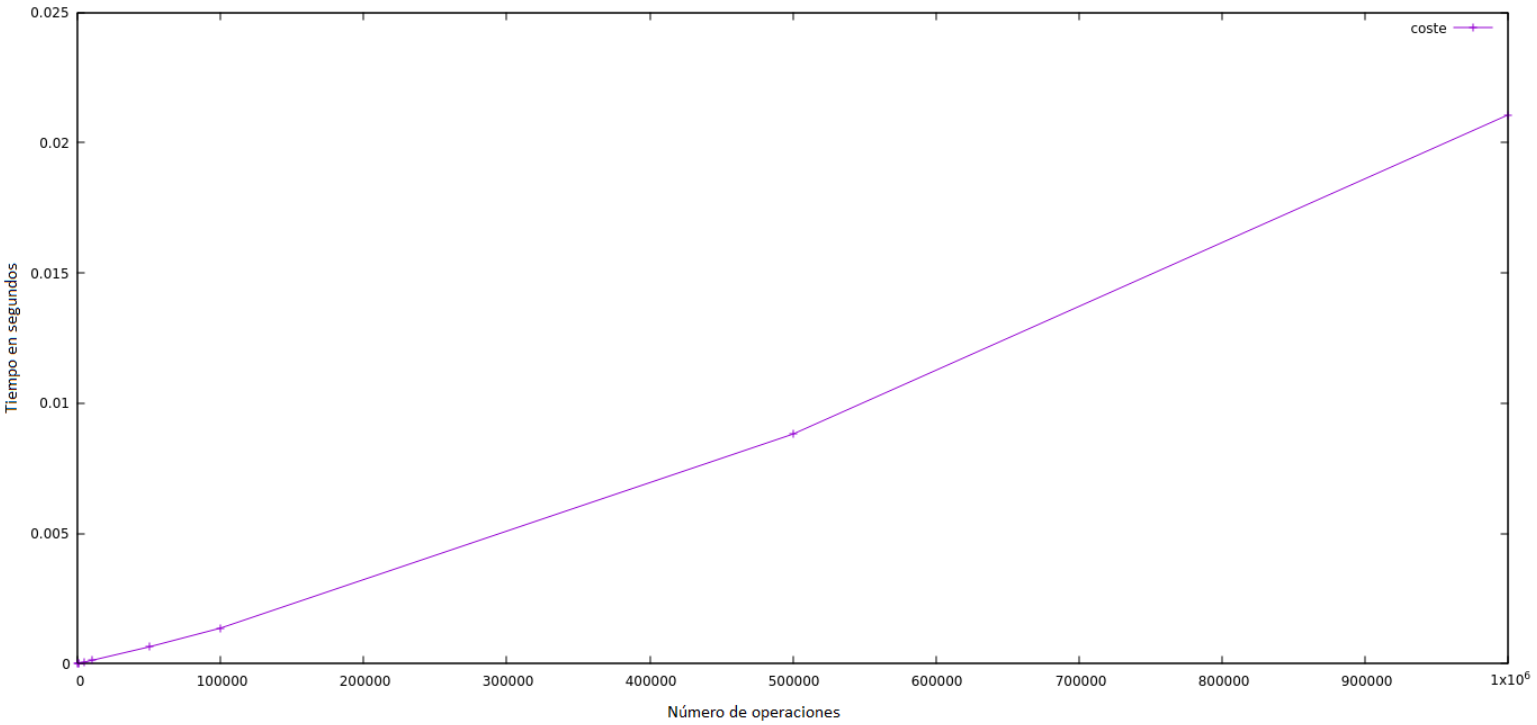


Gráfico 3: se puede encontrar en los archivos del programa como "todas.png"

3.4. Análisis de tiempos

Este tipo de árbol se trata de árboles binarios que, a diferencia de los AVL, no mantienen un invariante de equilibrio en la altura, por lo que en el caso peor puede llegar a tener costes de tiempo en O(n). No obstante se le puede atribuir un coste amortizado de O(logn) gracias a la función "flotación", que permite modificar la función de potencial de la estructura.

Se puede ver en la gráfica 1 y en la gráfica 3 como el coste de realizar n operaciones, siendo n un valor en un rango de 100 a 1.000.000, resulta en una gráfica que se asemeja bastante a una función n*log(n) (siendo n el nº de operaciones). En la gráfica 2, podría asemejarse a una función log(n) (siendo n el tamaño), sin embargo tiene "picos" ya que al no tener el invariante de altura, en algunos casos puede llegar a recorrer n elementos.

Estos resultados se han obtenido al insertar, borrar y buscar valores aleatorios, es decir por normal general no están en el caso peor. Con lo que concuerda con el resultado esperado al utilizar esta estructura en un caso general.