Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC2003 – Algoritmos y Estructura de Datos

Semestre II – 2015
Michel Ramírez – 14069
Cristian de León – 14015
Fernanda Dávila – 14482

HOJA DE TRABAJO 9

Implementación de mapeo y
Binary Search Trees (BST) autobalanceados

PRUEBAS UNITARIAS DE IMPLEMENTACIONES

Figura 1. Pruebas unitarias para tree1 (Splay Tree)

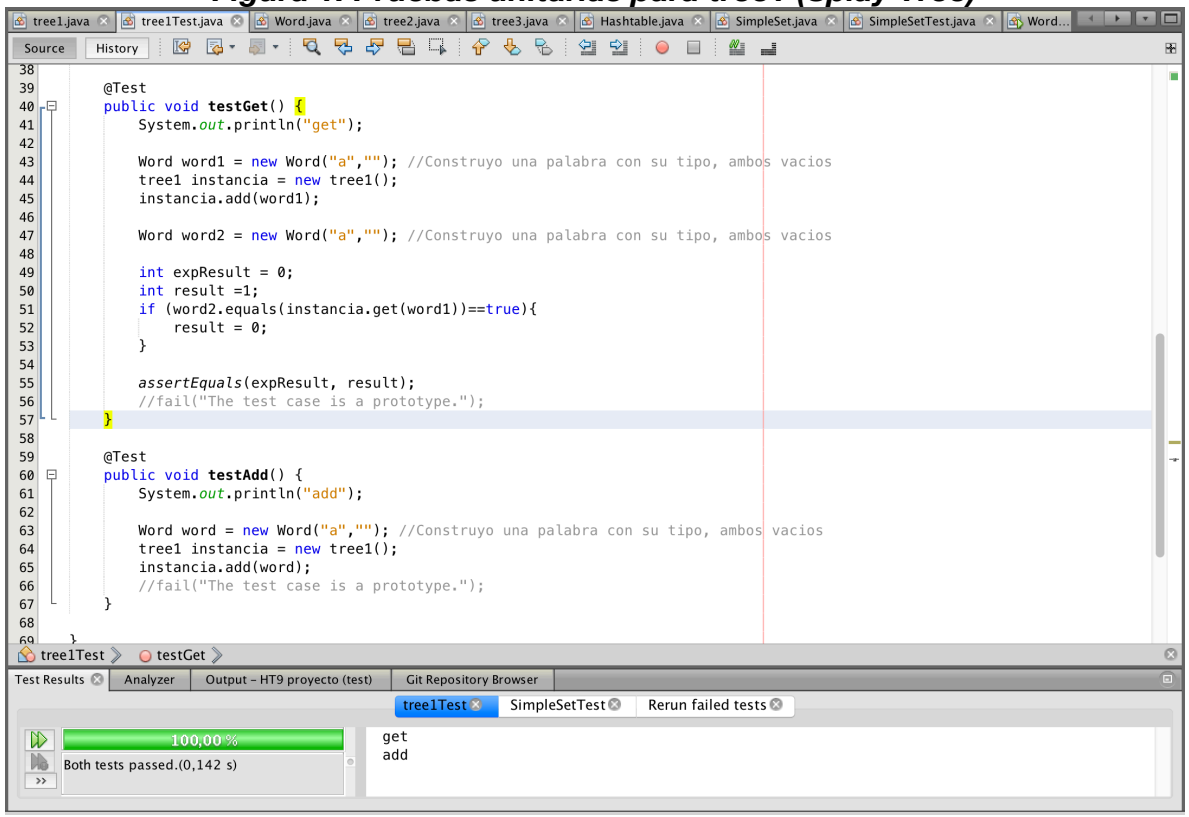


Figura 2. Pruebas unitarias para tree2 (Black Red Tree)

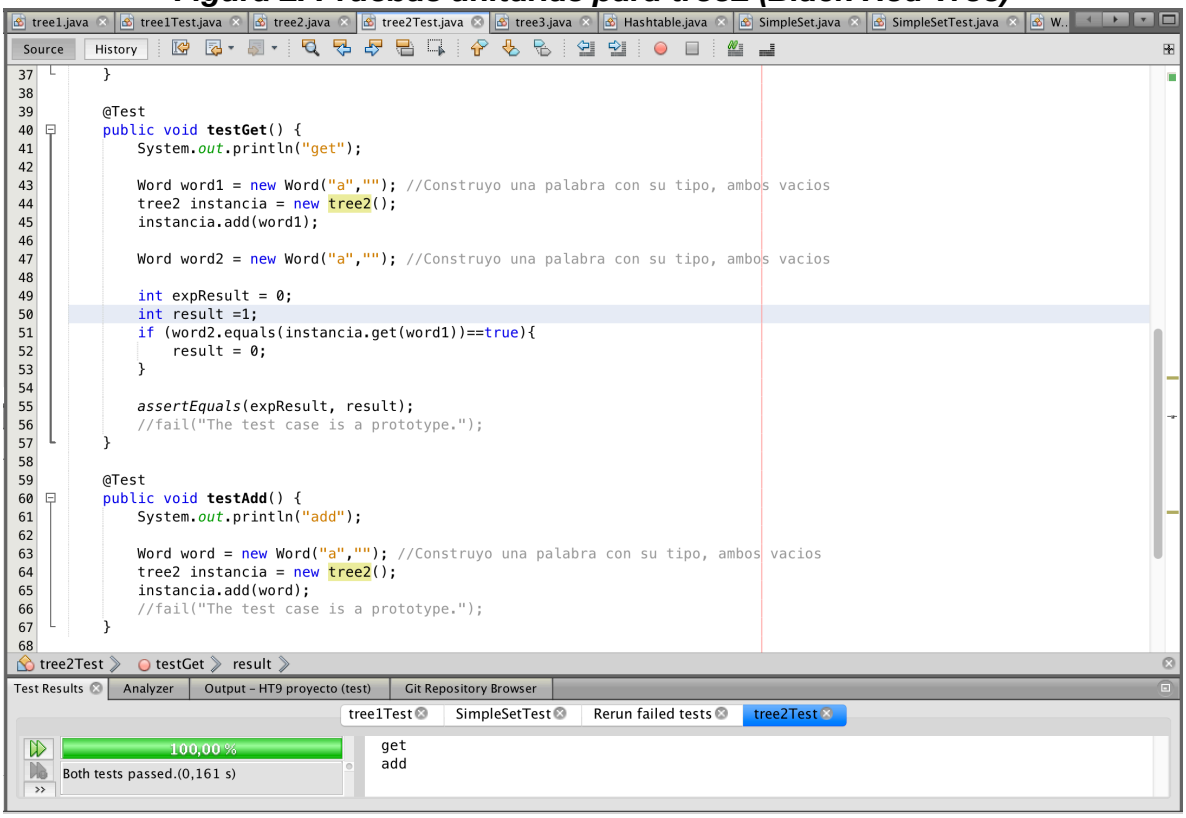


Figura 3. Pruebas unitarias para tree3 (TreeMap)

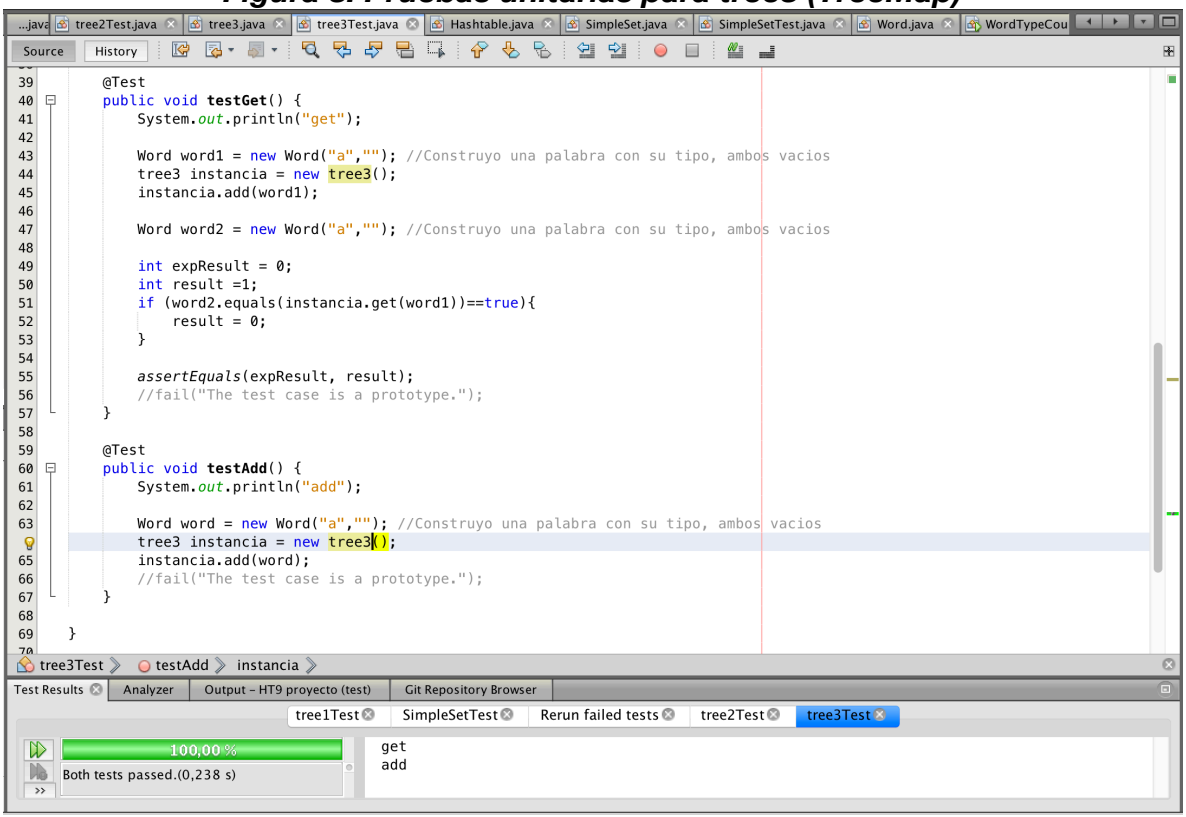


Figura 4. Pruebas unitarias para Hashtable (HashMap)

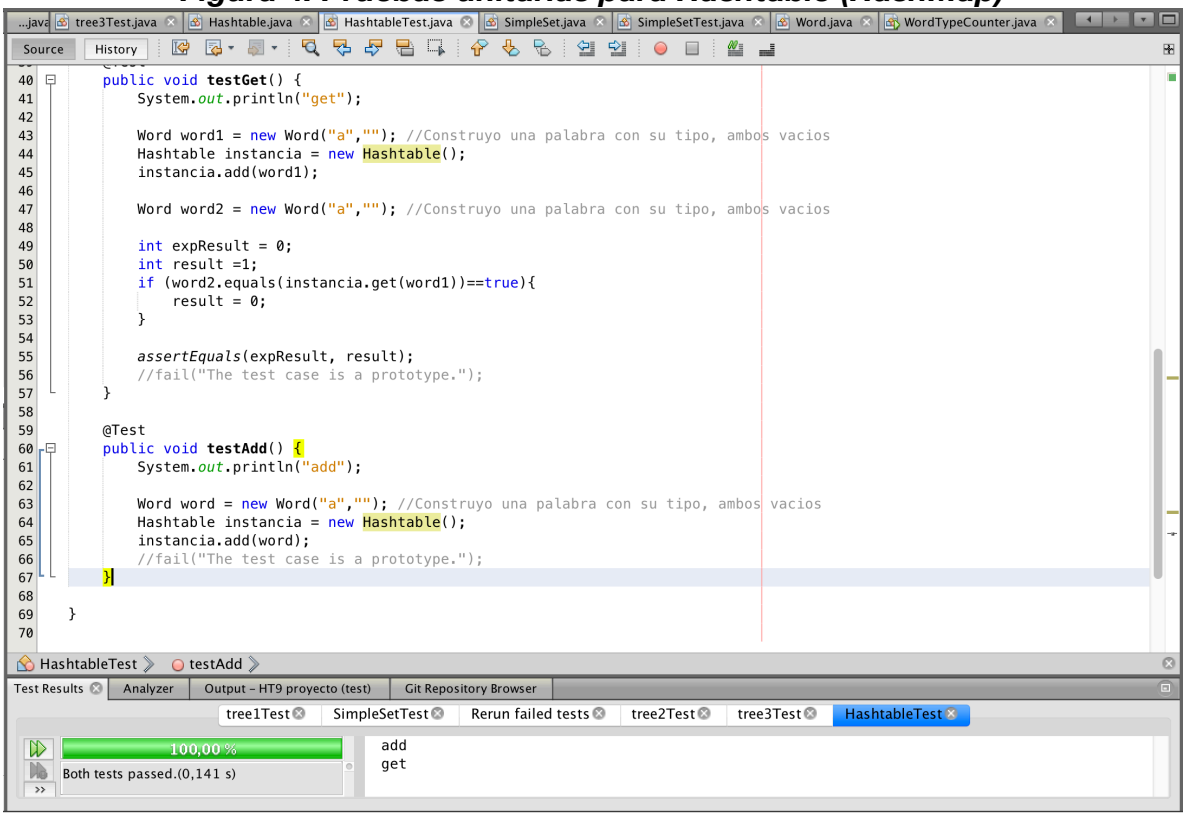
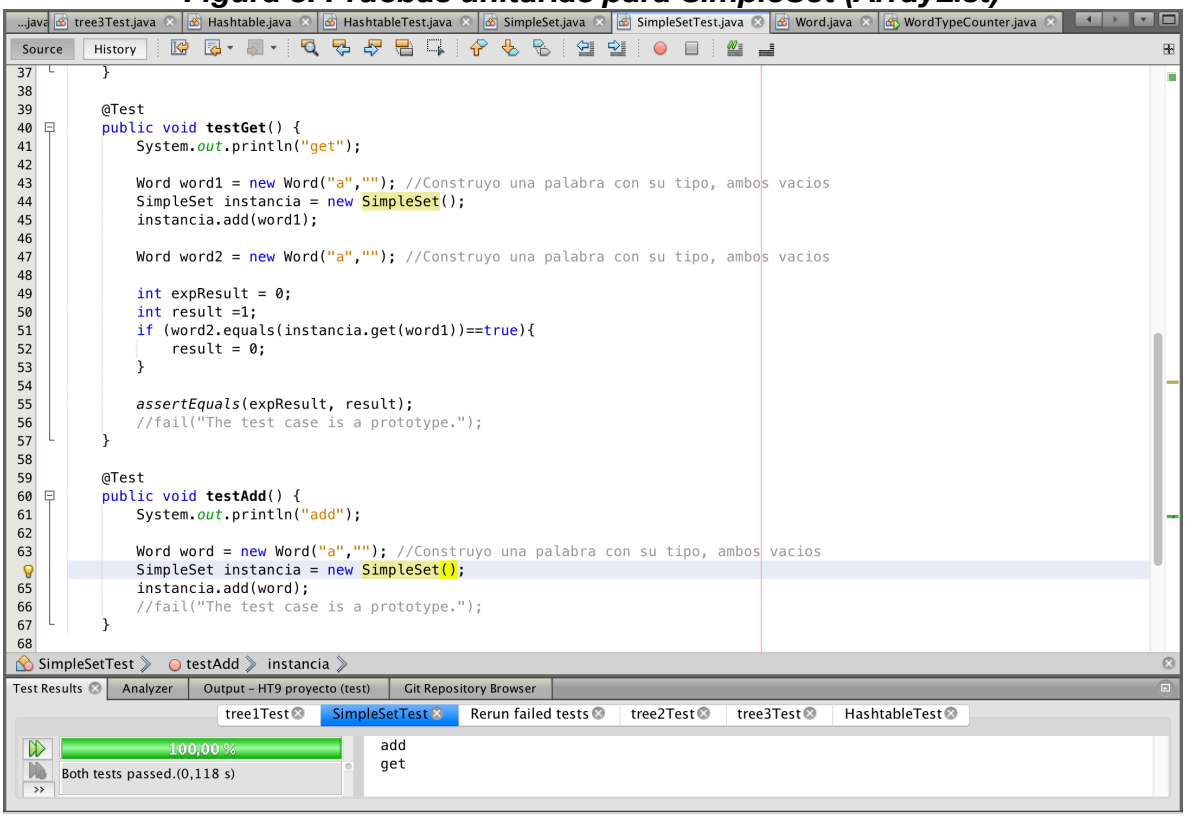


Figura 5. Pruebas unitarias para SimpleSet (ArrayList)



PROFILER DE IMPLEMENTACIONES

Figura 6. Profiler para SimpleSet (ArrayList)

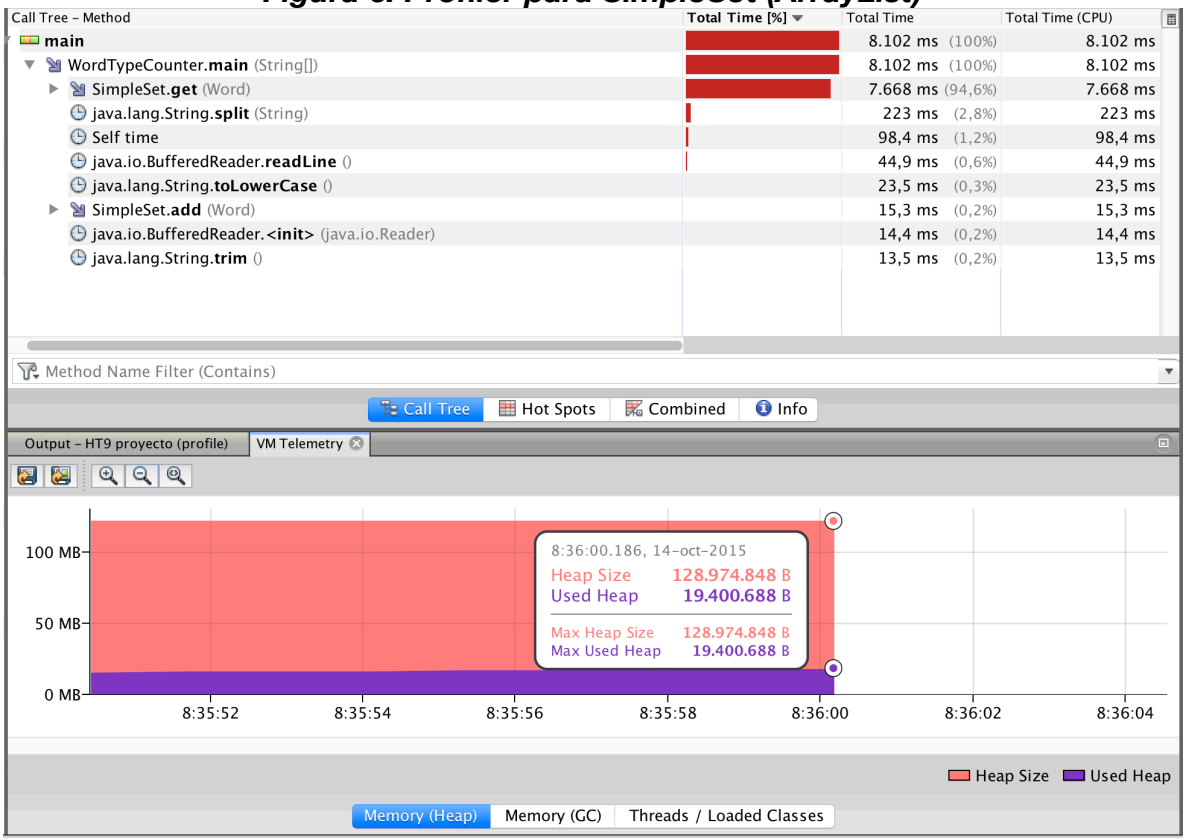


Figura 7. Profiler para tree2 (Black Red Tree)

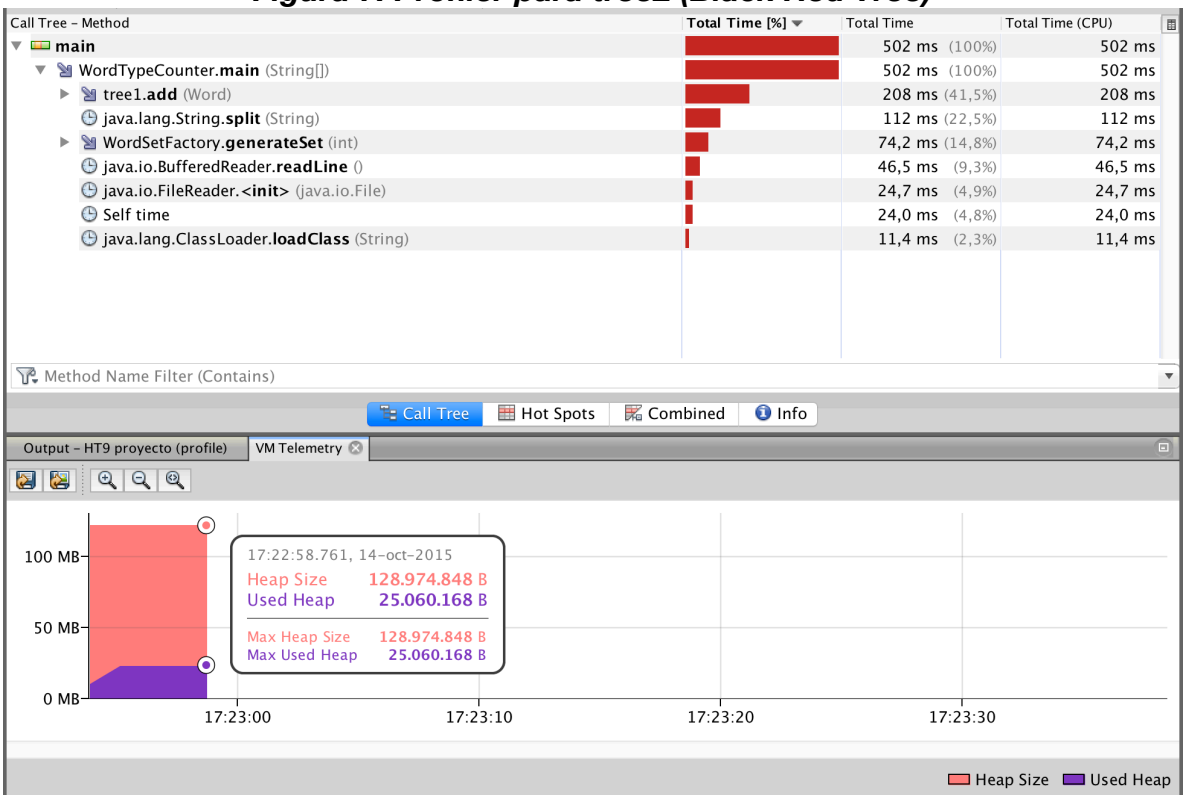


Figura 8. Profiler para tree1 (Splay Tree)

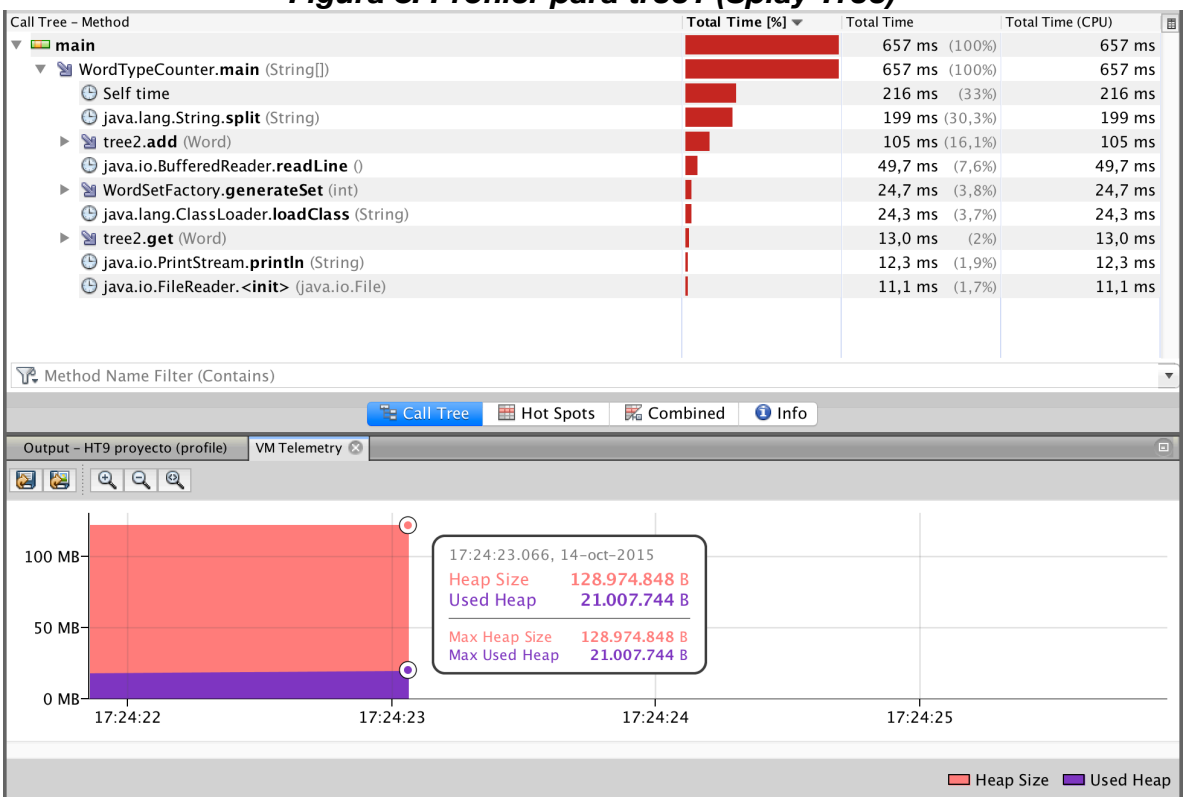


Figura 9. Profiler para HashTable (HashMap)

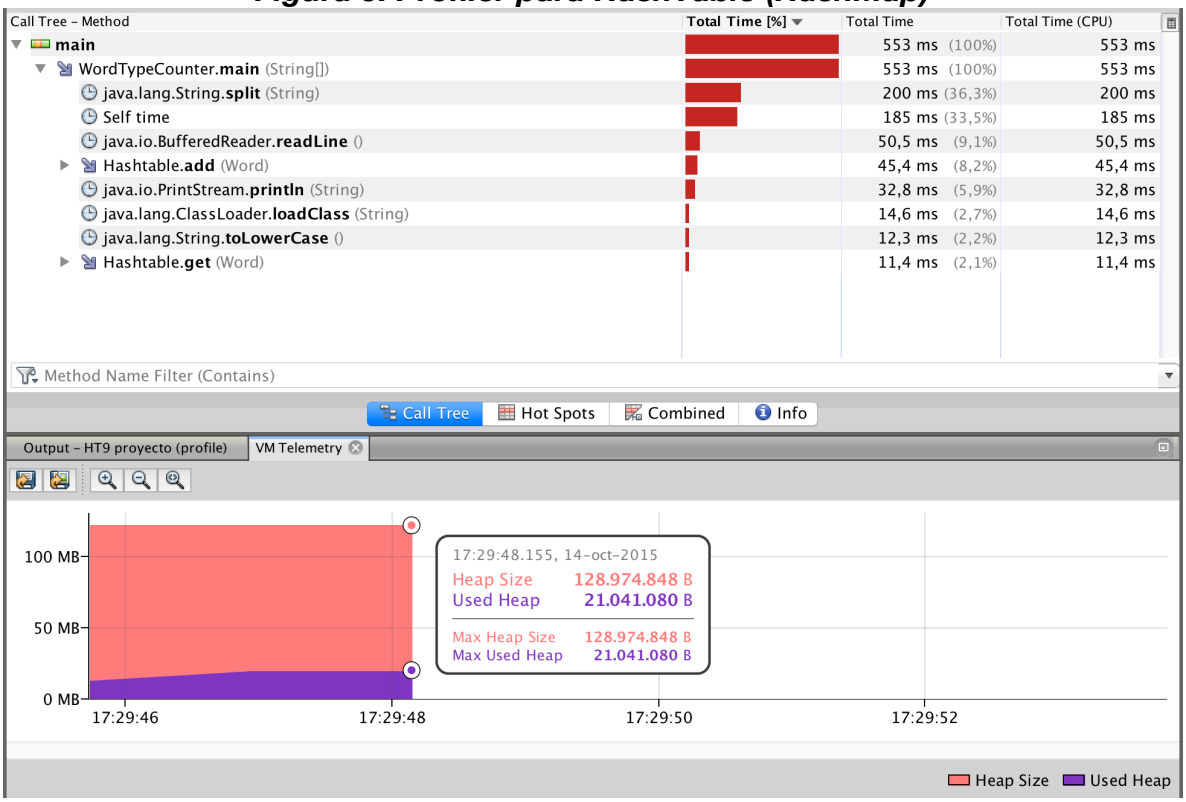
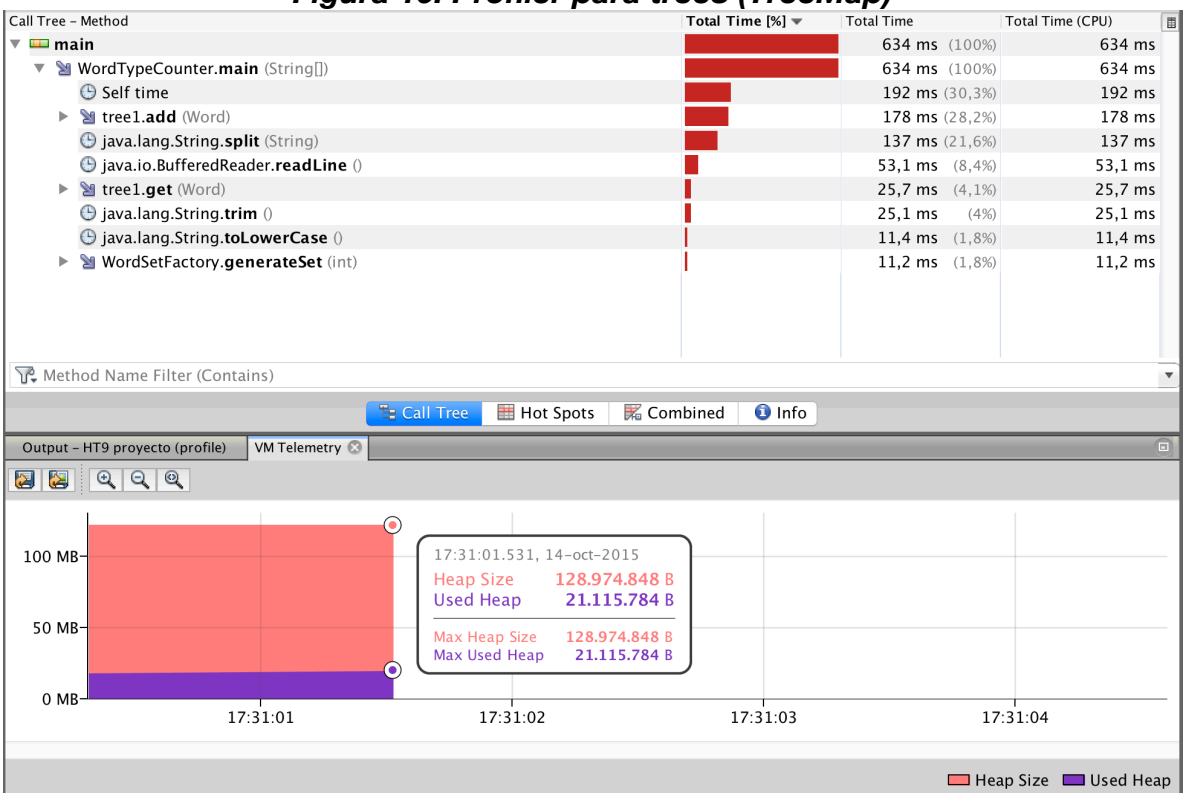


Figura 10. Profiler para tree3 (TreeMap)



DISCUSIÓN Y CONCLUSIÓN

RedBlackTree fue implementada en la clase tree2. El programa con esta estructura de datos fue el que **menos tiempo** tardó en llevar a cabo el programa. Al ver la figura 7, se puede observar que sólo tardó 502 ms en ejecutarse. El programa que demoró **más tiempo** fue SimpleSet, el cual estructuró los datos en un ArrayList; este demoró más de 8 mil milisegundos.

En cuanto al espacio de heap utilizado: el BlackRedTree fue la implementación que **más espacio** requirió. La estructura del ArrayList fue la que **menos espacio** usó. El método get() **más veloz** fue el de HashMap (11.4 ms); el get() **más lento** fue el de ArrayList (7668 ms). El método add() **más veloz** fue el de ArrayList (15.39 ms); el add() **más lento** fue el de BlackRedTree (208 ms).

Proponemos la implementación **RedBlackTree** como la más adecuada si el espacio de almacenamiento no es muy reducido. Si el almacenamiento es un problema, sugerimos utilizar la estructura de **HashTable**, pues ocupa 4.02 MB menos espacio de heap y es la segunda implementación más ágil.