

Hoja de trabajo 6 MAPS

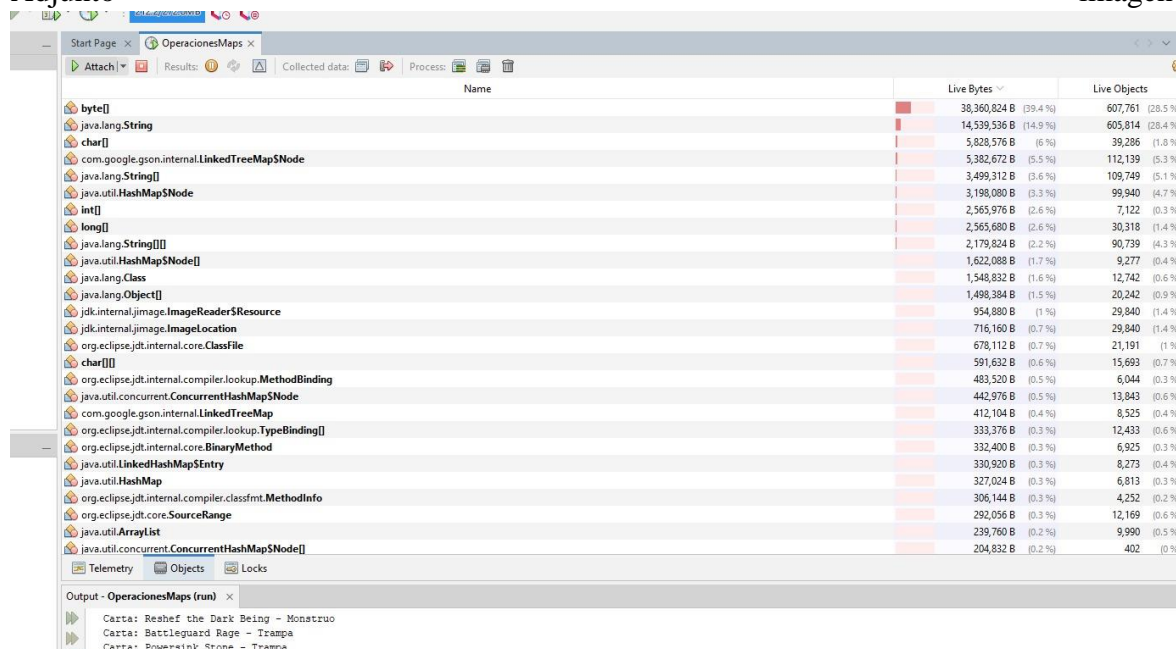
Link repositorio Github: <https://github.com/DiegoLinares11/HT6>

Anotaciones:

Yo programe en visual studio, sin embargo, para hacer los profilers lo hice con NetBeans lo cual tuve que cambiar mi programa para que me funcionara ahí, corrí, hice mis profilers y cuando los quería volver a ejecutar me decía que ya tenia unos profilers hechos y que topaban con los nuevos por lo que solo tengo una imagen de los profiles, sin embargo solo hablan en Bits y no en tiempos

Adjunto

imagen*



Name	Live Bytes	Live Objects
byte[]	38,360,824 B (39.4 %)	607,761 (28.5 %)
java.lang.String	14,539,536 B (14.9 %)	605,814 (28.4 %)
char[]	5,828,576 B (6 %)	39,286 (1.8 %)
com.google.gson.internal.LinkedTreeMap\$Node	5,382,672 B (5.5 %)	112,139 (5.3 %)
java.lang.String[]	3,499,312 B (3.6 %)	109,749 (5.1 %)
java.util.HashMap\$Node	3,198,080 B (3.3 %)	99,940 (4.7 %)
int[]	2,565,976 B (2.6 %)	7,122 (0.3 %)
long[]	2,565,680 B (2.6 %)	30,318 (1.4 %)
java.lang.String[][]	2,179,824 B (2.2 %)	90,739 (4.3 %)
java.util.HashMap\$Node[]	1,622,088 B (1.7 %)	9,277 (0.4 %)
java.lang.Class	1,548,832 B (1.6 %)	12,742 (0.6 %)
java.lang.Object[]	1,498,384 B (1.5 %)	20,242 (0.9 %)
jdk.internal.image.ImageReader\$Resource	954,880 B (1 %)	29,840 (1.4 %)
jdk.internal.image.ImageLocation	716,160 B (0.7 %)	29,840 (1.4 %)
org.eclipse.jdt.internal.core.ClassFile	678,112 B (0.7 %)	21,191 (1 %)
char[][]	591,632 B (0.6 %)	15,693 (0.7 %)
org.eclipse.jdt.internal.compiler.lookup.MethodBinding	483,520 B (0.5 %)	6,044 (0.3 %)
java.util.concurrent.ConcurrentHashMap\$Node	442,976 B (0.5 %)	13,843 (0.6 %)
com.google.gson.internal.LinkedTreeMap	412,104 B (0.4 %)	8,525 (0.4 %)
org.eclipse.jdt.internal.compiler.lookup.TypeBinding[]	333,376 B (0.3 %)	12,433 (0.6 %)
org.eclipse.jdt.internal.core.BinaryMethod	332,400 B (0.3 %)	6,925 (0.3 %)
java.util.LinkedHashMap\$Entry	330,920 B (0.3 %)	8,273 (0.4 %)
java.util.HashMap	327,024 B (0.3 %)	6,813 (0.3 %)
org.eclipse.jdt.internal.compiler.classfmt.MethodInfo	306,144 B (0.3 %)	4,252 (0.2 %)
org.eclipse.jdt.core.SourceRange	292,056 B (0.3 %)	12,169 (0.6 %)
java.util.ArrayList	239,760 B (0.2 %)	9,990 (0.5 %)
java.util.concurrent.ConcurrentHashMap\$Node[]	204,832 B (0.2 %)	402 (0 %)

Por dicho inconveniente me di a la tarea de investigar las complejidades de estos 3 tipos de mapas.

1. HashMap: Complejidad: el tiempo de inserción, eliminación y búsqueda en un HashMap es en promedio $O(1)$ en el caso óptimo. Sin embargo, si hay muchas colisiones (es decir, entradas con el mismo hash), el tiempo de búsqueda puede ser $O(n)$, donde n es el número de entradas.
2. TreeMap: Complejidad: el tiempo de inserción, eliminación y búsqueda en un TreeMap es $O(\log n)$, donde n es el número de entradas.
3. LinkedHashMap: Complejidad: el tiempo de inserción, eliminación y búsqueda en un LinkedHashMap es $O(1)$ en el caso óptimo, igual que un HashMap. Además, también

Universidad del Valle de Guatemala

Algoritmos y Estructuras de Datos

Hoja de trabajo 6

se puede iterar por los datos en orden de inserción en $O(n)$ tiempo, donde n es el número de entradas.