Data Structure for efficient indexing of files

Mateo Restrepo Sierra
Nicolas Restrepo López
Medellín, October 30th, 2017

Data Structure design

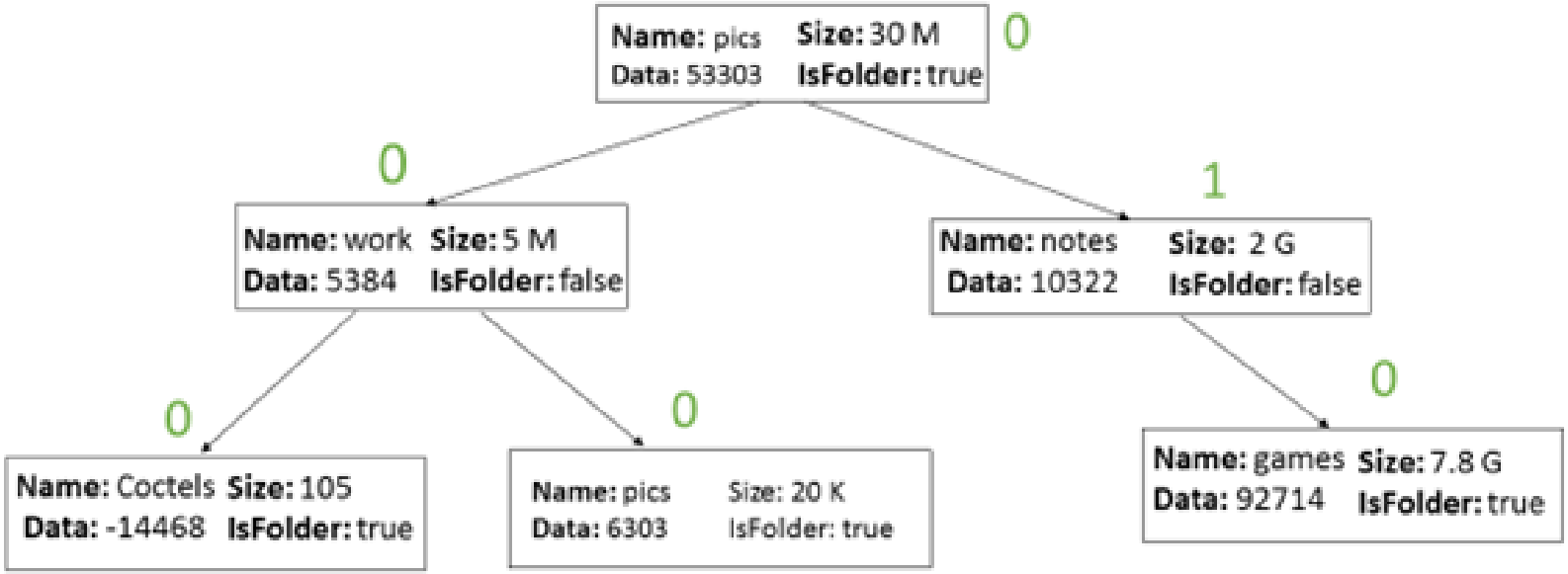
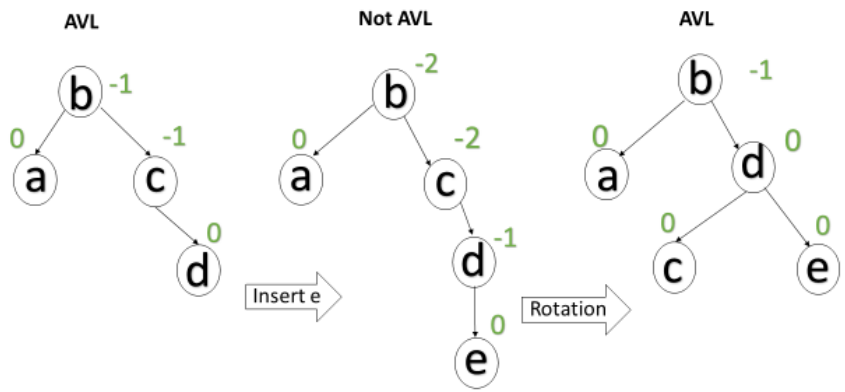


Figure 1: This is an early implementation of our AVL Tree. The numbers above the tree are the balanced factor. It does not have all the fields that the objects have.

Data Structure Operations



Method	Complexity
Insert a Node	O(Log n)
Delete a Node	O(Log n)
Search an Element	O(Log n)
Print Pre-Order	O(n)
Print In Order	O(n)
Print Post Order	O(n)

Figure 2: Image of an operation of insertion.

Figure 3: Complexity analysis for AVL Tree standard operations.

Design Criteria of the Data Structure

- To solve the problem, we should used, extensively, the operation of searching
- Trees are the most ideal to solve this problem
- Searching has a time complexity of $O(\log n)$
- The AVL Tree is one of the most efficient Data Structure, because it has shorter height in the subtrees than other Self-Balancing Trees like the Red-Black Tree
- AVL Tree provides faster look-ups
- Hash functions were used for insertion

Time and Memory Consumption

Running Times (Milliseconds)			
	ejemplito.txt	**treeEtc.txt**	**juegos.txt**
Insert	15	184	3236
Search Directory	~0	~0	~0
Search Sub directories	~0	~0	335,7
Search Low	1215	4343	15440
Search High	~0	~0	~0
Search Range	~0	~0	1117

Figure 4: Execution time in milliseconds by three different .txt files.

Memory Used (Megabytes)			
	ejemplito.txt	**treeEtc.txt**	**juegos.txt**
First Try	9,566	13,750	48,253
Second Try	9,565	13,749	47,595
Third Try	9,565	13,750	52,018

Figure 5: Memory use in Megabytes by three different .txt files.

Implementation

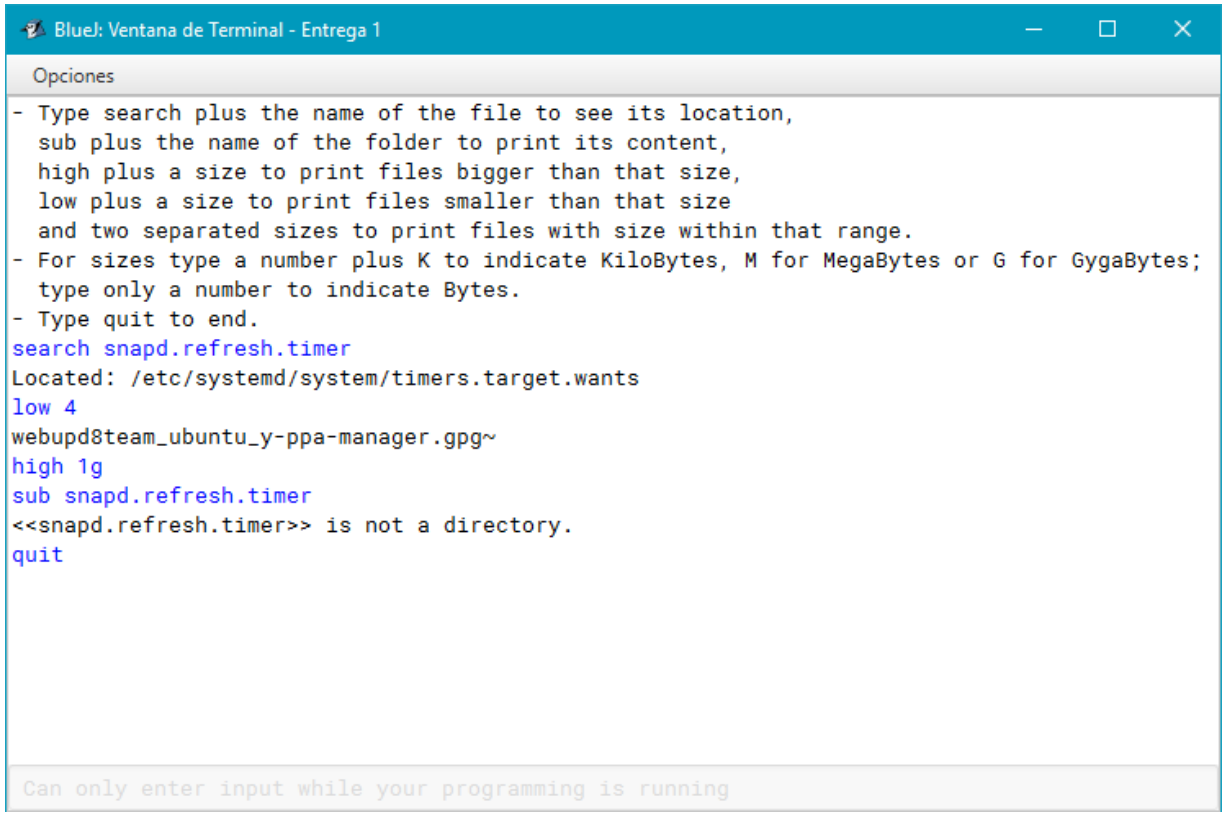


Figure 6: Our implementation running in BlueJ for treeEtc.txt.