WHW 3-1

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

WHW 3-2

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명

WHW 3-3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | AVL tree | T-tree | | |
|  | O(log2n) | O(log2n/2) | O(log2n/5) | O(log2n/10) |
| 100 | 6.64 | 5.64 | 4.32 | 3.32 |
| 1,000 | 9.966 | 8.966 | 7.644 | 6.644 |
| 100,000 | 16.61 | 15.61 | 14.29 | 13.29 |

Insertion and deletion time complexity of AVL tree is O(log n). And Insertion and deletion time complexity of T-tree is O(log n/m). As you can see the following table, T-tree is better in terms of performance unless m(number of keys per node) equals to 1.

Let’s say k is the memory of data you store and l is the memory of address of the nodes. nk + 3nl/m bytes are for T-tree and nk + 2nl bytes are for AVL tree. If m equals to 1, AVL tree is better in terms of memory requirements. But if m is bigger than 1, T-tree is better in terms of memory requirements.

Let’s look at insert/delete processing overhead now. time is mainly spent on rotation. AVL tree must be re-balanced by performing “tree rotations” around the node whose balance factor becomes 2 or -2. Also in T-tree case, you should perform tree rotations if tree becomes out of balance (the balance factor is any node becomes +2 or -2). And you should merge or split 2 nodes if overflows or underflow occurs. Rotation of AVL tree occurs more often.

In summary, insert/delete processing overhead of T-tree is reduced.