**Data Structures**

**Spring 2020**

**Written Homework #4**

**Date: 2020. 04. 29**

**ID/Name: 201533661 이승수**

**[HW#4-1]**

**Construct a 2-3 Tree with keys D, A1, T1, A2, S, T2, R1, U1, C, T3, U2, R2, E**

**in the given order, starting from an empty tree.**

**-(you must show each insert and each node split)**

**From the constructed 2-3 Tree, delete the nodes with keys A1, T1, T2, T3**

**in the given order.**

**-(you must show each delete and each node merge) (20 points)**

**Answer)**

**Start :** 

**[Insertion Phase]**

|  |  |
| --- | --- |
| Insert D | Insert A1 |
|  |  |

|  |
| --- |
| Insert T1 |
|  |

|  |  |
| --- | --- |
| Insert A2 | Insert S |
|  |  |

|  |
| --- |
| Insert T2 |
|  |

|  |  |
| --- | --- |
| Insert R1 | Insert U1 |
|  |  |

|  |
| --- |
| Insert C |
|  |

|  |
| --- |
| Insert T3 |
|  |

|  |
| --- |
| Insert U2 |
|  |

|  |
| --- |
| Insert R2 |
|  |

|  |
| --- |
| Insert E |
|  |

**[Deletion Phase]**

|  |
| --- |
| Delete A1 |
|  |

|  |
| --- |
| Delete T1 |
|  |

|  |
| --- |
| Delete T2 |
|  |

|  |
| --- |
| Delete T3 |
|  |

**[HW#4-2]**

**Construct a T-Tree (where M=2) with keys 20, 80, 60, 40, 15, 25, 30, 35**

**-in the given order, starting from an empty tree. (14 points)**

**-(you must show each insert; node split and tree rotation)**

**From the constructed T-Tree, delete the nodes with keys 20, 35, 60, 80**

**in the given order.**

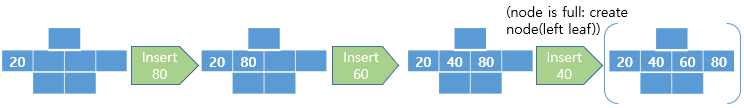
**you must show each delete; node merge and tree rotation)**

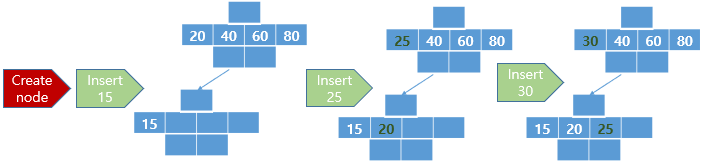
**Answer)**

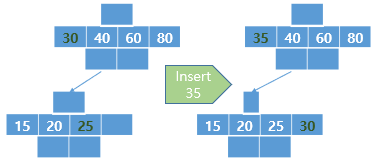
**Start:**

**[Insertion Phase]**

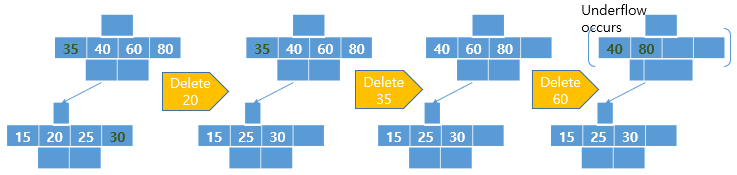


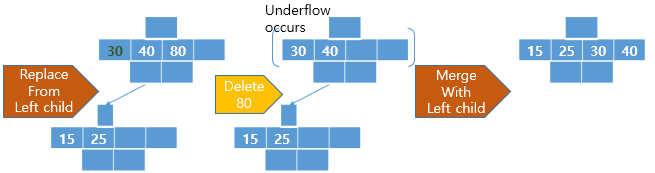






**[Deletion Phase]**





**[HW#4-3]**

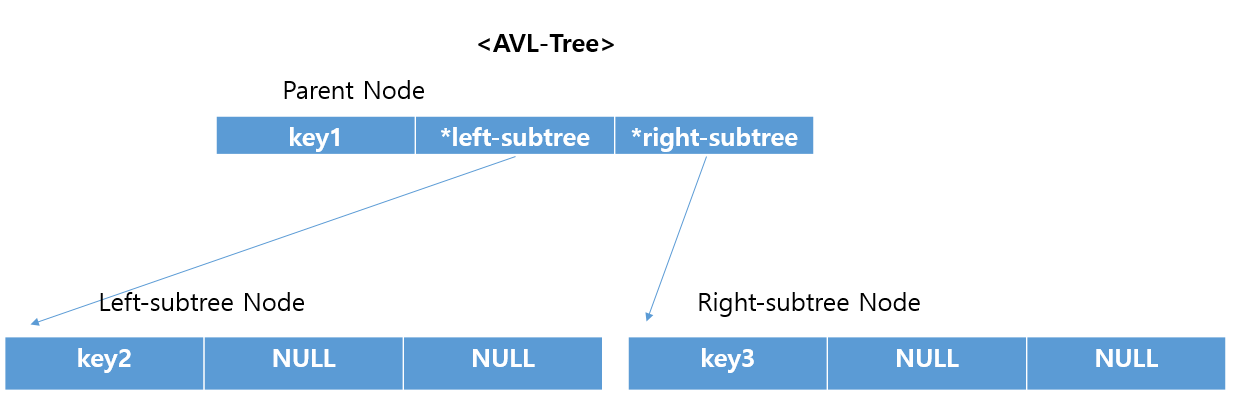
**Discuss the tradeoffs between an AVL tree and a T tree. (Need trend and summary comparison) (6 points)**

**(hint: Compare the two in terms of performance, memory requirements, and insert/delete processing overhead; work with N = 100, 1000, 100000, for example)**

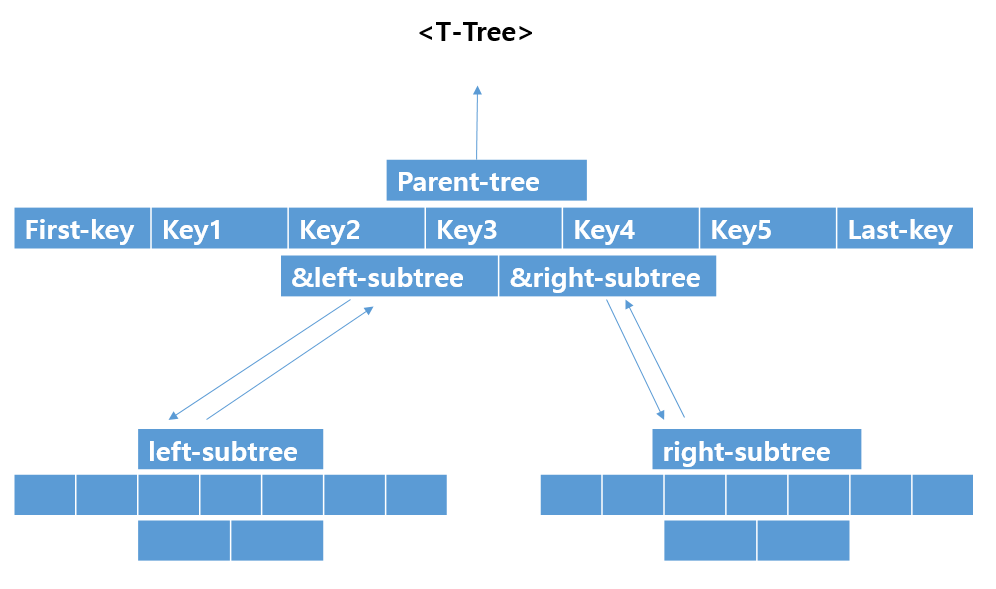
**Answer)**

Before discussing about the Tradeoffs, we need to know simply about character of AVL-tree and T-tree.

AVL-tree is type of binary tree structure that maintain Balance Factor of left-subtree and right-subtree. It need to be sorted the tree whenever node is inserted or deleted. Difference between left&right subtree must be maintained less than 1.



T-tree is type of binary tree structure that store bundles of data in each node. When searching at T-tree, firstly check between the range of first and last key of the T-tree node. If the key is out of the range of T-tree node,



In term of performance, T-tree is useful for searching for key among the large amount of data. For example, pretend to search for particular key when N=100,000. We can firstly search for the key by just comparing it with first and last key of each nodes. If the key is caught in the range, search for the key in the node and else, just pass the node and look for another T-tree nodes. As a result, T-tree can probabilistically save time at searching from large amount of data. Calculate the reduced tree height, T-tree can reduce log2(25,000)(=log2(100,000/4)) height when M is 4.

However, sometimes, worst case can be occurred, so searching time can be cost much more time than AVL-tree.

In term of memory requirements, more efficient case tree structure is different in case by case. AVL-tree node structure needs 3 data(key, &left-child, &right-child) and T-tree node structure needs M+5 data(pointer to parent node, key1~keyM, first\_key, last\_key, &left-child node, &right-child node). When there are N keys, AVL-tree needs 3\*N data memories and T-tree needs (N/M)\*(M+5) data memories. As a result, when M is bigger than 2.5, T-tree structure is efficient in term of memory requirements.

In term of overhead at insert/delete processing, we think usually T-tree structure is efficient because of rotations at AVL-tree. When inserted/deleted datas are sorted in ascending order or descending order, T-tree structure don’t need many rotation but AVL-tree structure needs numerous node rotations even inserted in ascending order. As data become larger, AVL-tree will be more inefficient because of overhead.