B-Trees

ΜΠΟΜΠΟΤΑΣ ΑΓΟΡΑΚΗΣ

mpompotas@ceid.upatras.gr

Εισαγωγή

B-tree:

- Δενδρική δομή
- Τα δεδομένα σε ένα B-tree αποθηκεύονται ταξινομημένα
- Χρησιμοποιείται σε υλοποιήσεις Βάσεων Δεδομένων και Filesystems

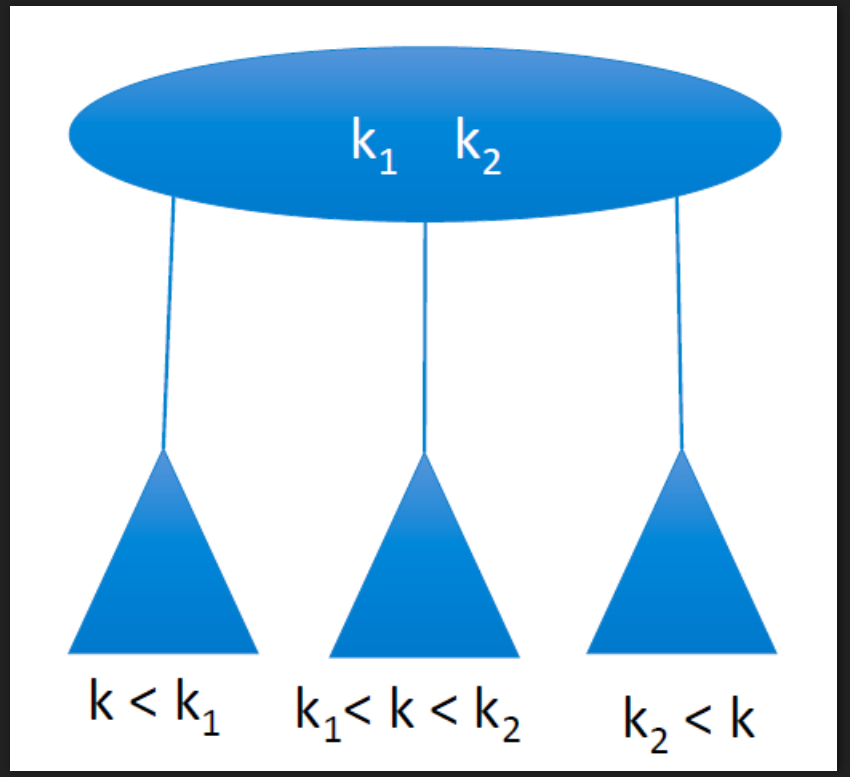
Ορισμός B-tree

Ένα B-tree τάξεως m, είναι ένα δέντρο με τα εξής χαρακτηριστικά:

- Η ρίζα έχει τουλάχιστον 2 παιδιά, εκτός εάν είναι φύλλο
- Κάθε κόμβος έχει το πολύ m παιδιά και m-1 κλειδιά
- Κάθε κόμβος πλην της ρίζας και των φύλλων έχουν τουλάχιστον $^{m}/_{2}$ παιδιά
- Όλα τα φύλλα έχουν το ίδιο ύψος
- Ένας εσωτερικός κόμβος με k παιδιά έχει ακριβώς k-1 κλειδιά

Ορισμός B-tree

- Τα κλειδιά οργανώνονται σε υποδέντρα παρόμοια με τον τρόπο που οργανώνονται στα δυαδικά δέντρα.



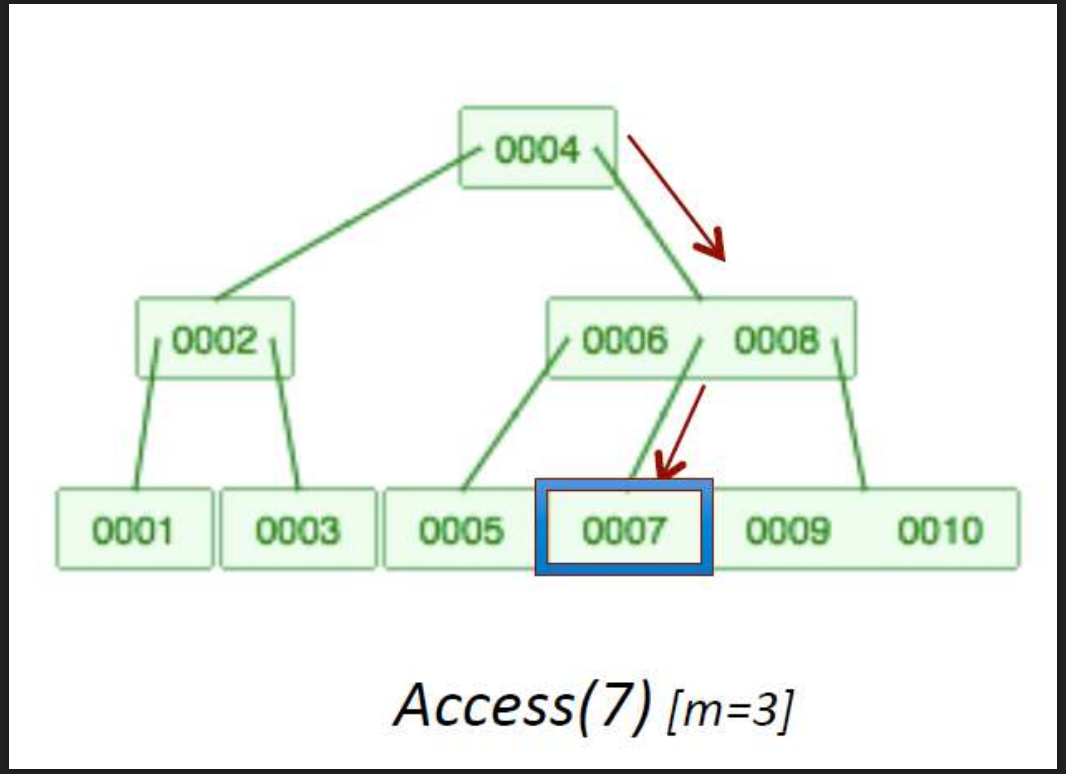
Πράξεις σε B-trees

Μας ενδιαφέρουν 3 πράξεις:

- Αναζήτηση (Access)
- Ένθεση (Insert)
- Διαγραφή (Delete)

Αναζήτηση (Access)

- Η αναζήτηση του στοιχείου x σε ένα B-tree ξεκινά από τη ρίζα και σε κάθε επίπεδο επιλέγεται το αντίστοιχο υποδένδρο στο οποίο περιέχεται η τιμή που αναζητείται.



Ένθεση (Insertion)

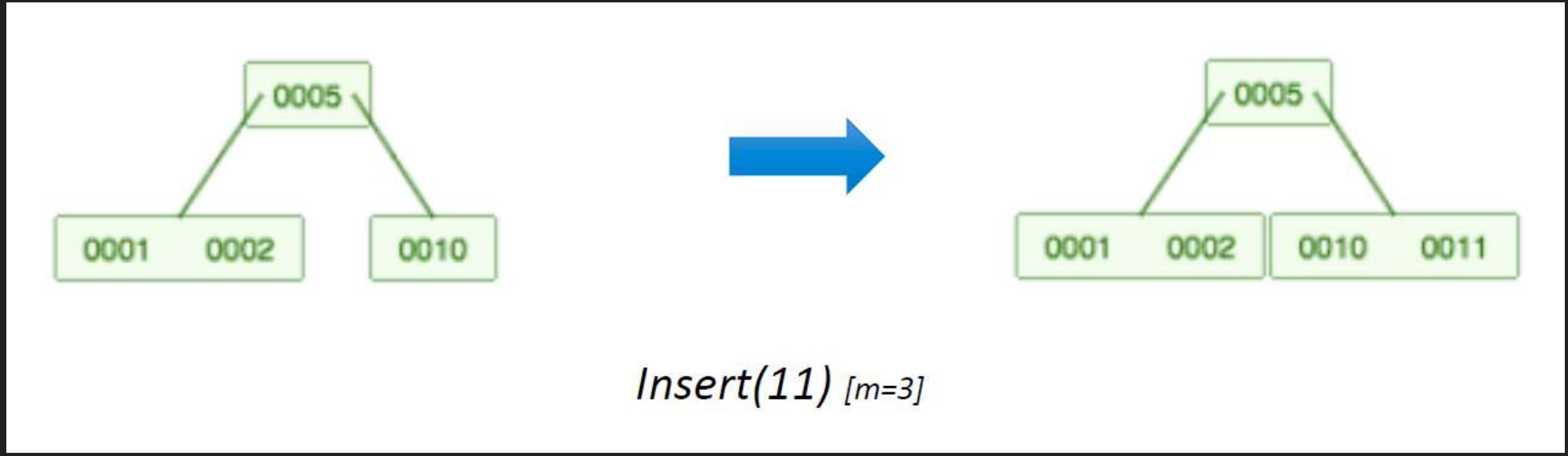
Για την εισαγωγή του στοιχείου x σε B-tree ακολουθούνται τα εξής βήματα:

- Εύρεση του κατάλληλου φύλλου y στο δέντρο για την εισαγωγή της τιμής x (Κλήση της Access(x)).
- Εισαγωγή του νέου κλειδιού στο φύλλο.
- Εφαρμογή επανορθωτικών πράξεων αν προκύπτει υπερχείλιση.

Ένθεση (Insertion)

Περίπτωση 1: Καμία Υπερχείλιση

Εάν δεν υπάρχει υπερχείλιση, η ένθεση γίνεται κανονικά.

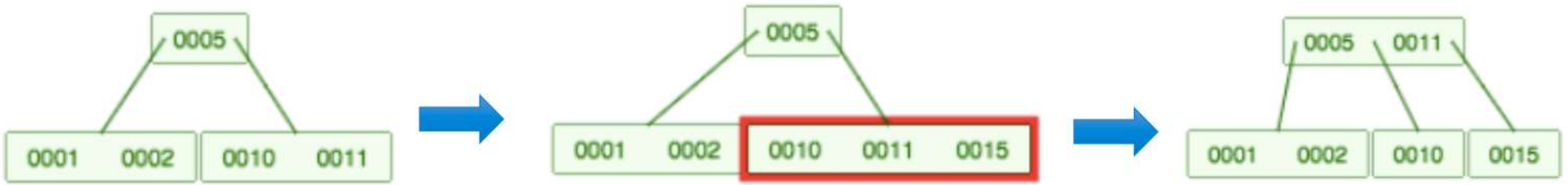


Ένθεση (Insertion)

Περίπτωση 2: Υπερχείλιση

Υπερχείλιση προκύπτει όταν μετά την ένθεση ένα φύλλο έχει m κλειδιά. Για να επιλυθεί το πρόβλημα εφαρμόζουμε τον μετασχηματισμό της διάσπασης ως εξής:

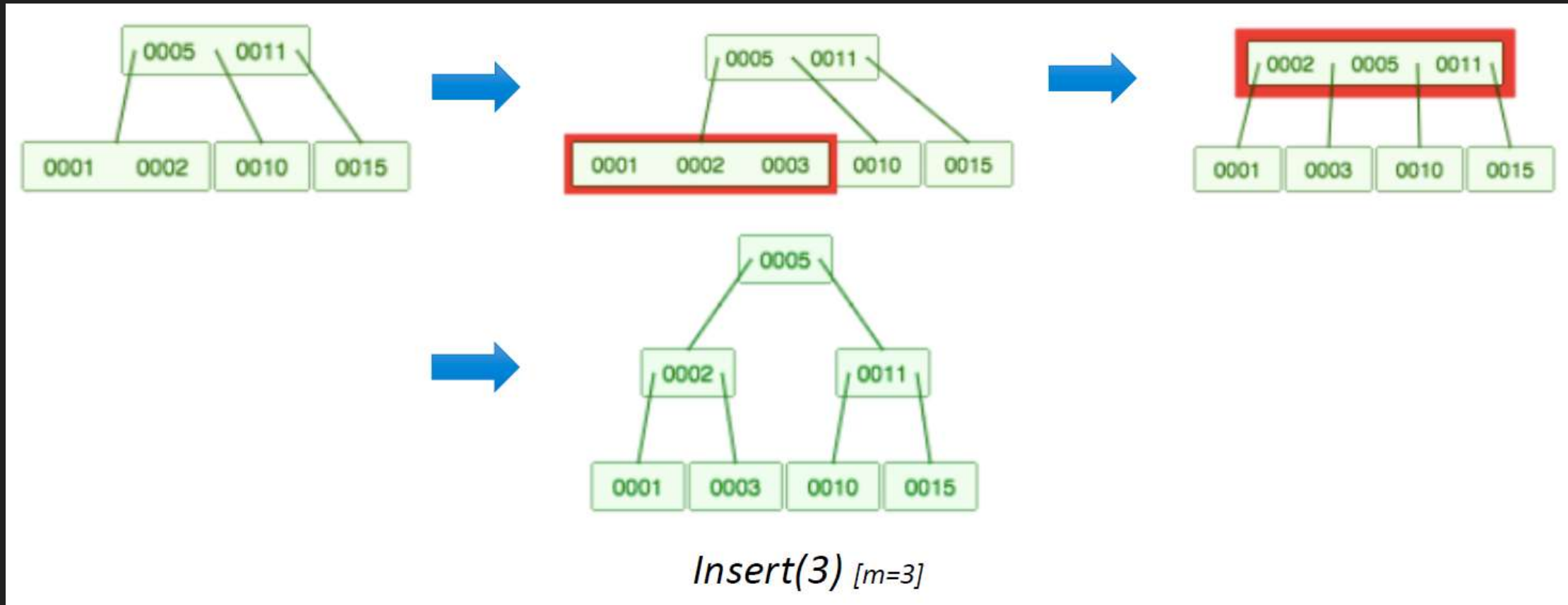
- Επιλέγεται το μεσαίο κλειδί x του κόμβου που είναι προς διάσπαση, μεταφέρεται στον γονέα, και ο κόμβος διασπάται σε 2 κόμβους.
- Τα μικρότερα κλειδιά από το x τοποθετούνται στο αριστερό υποδέντρο του x, ενώ τα μεγαλύτερα στο δεξιό.



Insert(15) [m=3]

Ένθεση (Insertion)

Μετά τη διάσπαση ενός φύλλου, μπορεί να έχουμε διαδοχικές διασπάσεις μέχρι και τη ρίζα.



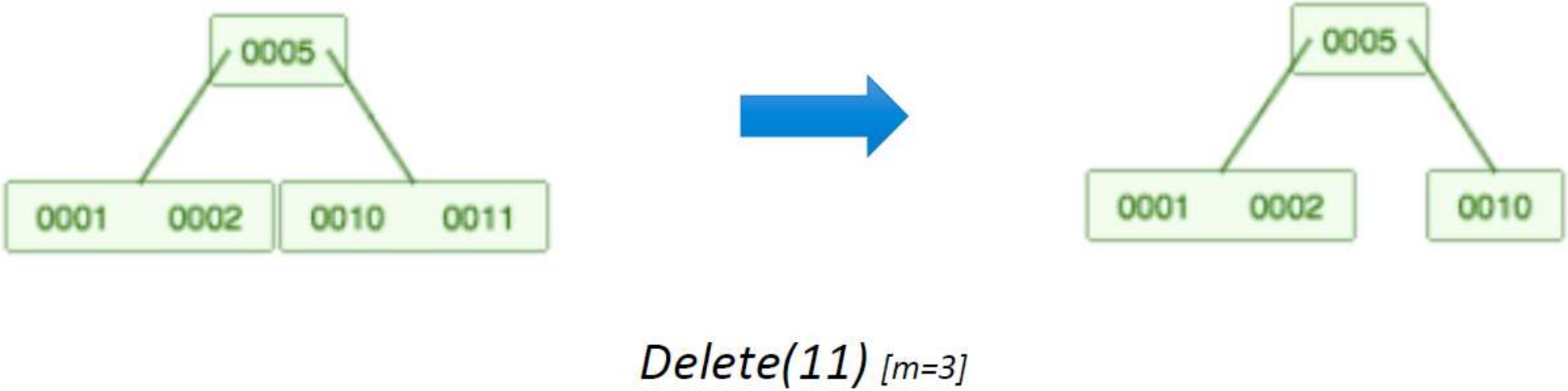
Διαγραφή (Deletion)

Για την διαγραφή ενός στοιχείου x σε B-tree ακολουθούνται τα εξής βήματα:

- Εύρεση του x σε ένα φύλλο του δέντρου (Κλήση της Access(x)).
- Διαγραφή του κλειδιού x
- Εφαρμογή επανορθωτικών πράξεων αν διαταραχθούν οι ιδιότητες του δέντρου.

Διαγραφή (Deletion)

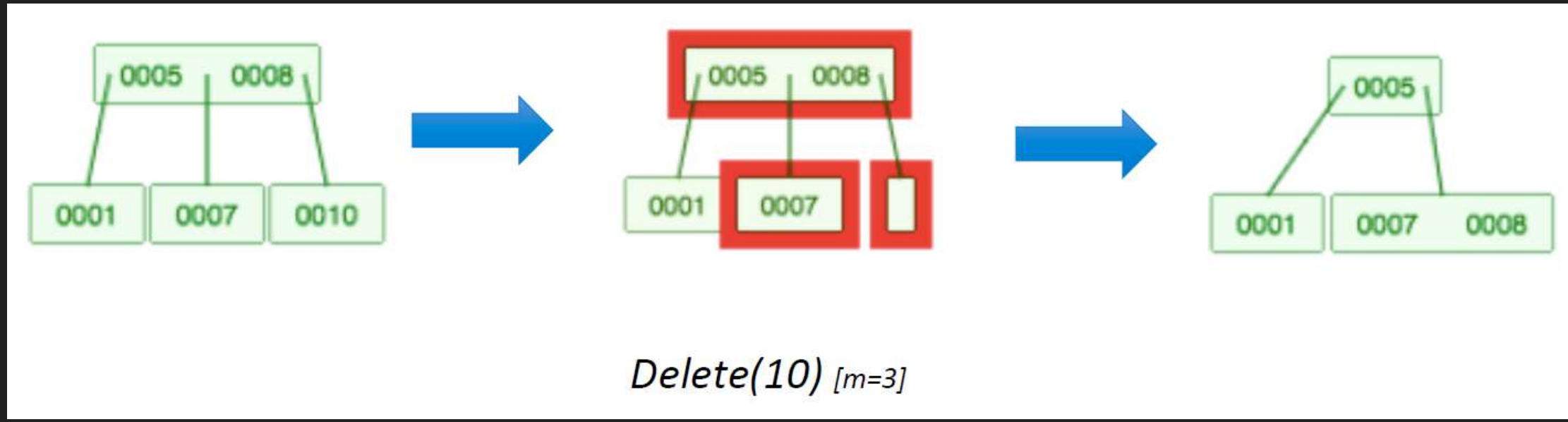
Περίπτωση 1: Η μορφή του δέντρου δε διαταράσσεται



Διαγραφή (Deletion)

Περίπτωση 3: Η μορφή του δέντρου διαταράσσεται, και ΔΕΝ υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Συγχώνευση



Απόδοση

Πολυπλοκότητες χειρότερης περίπτωσης:

- Search: O(log n)
- Insert: O(log n)
- Delete: O(log n)

Παρουσίαση Υλοποίησης

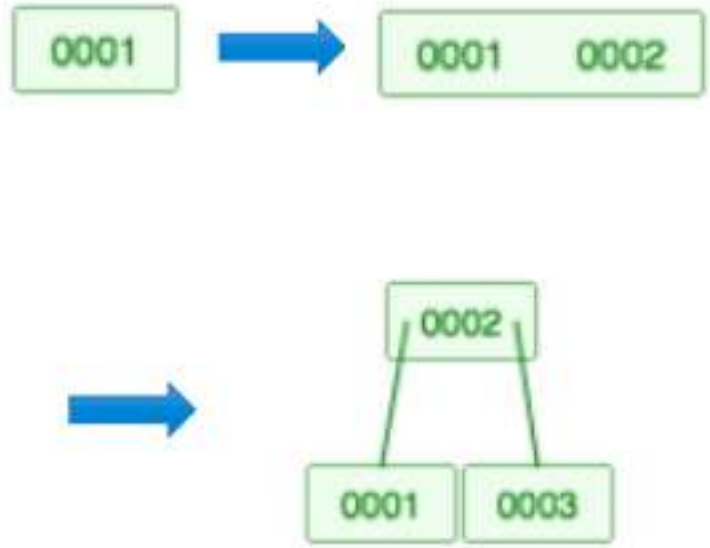
```
public class BTree<Key extends Comparable<Key>, Value>  {
    private static final int M = 3;    // max children per B-tree node = M-1

    private Node root;             // root of the B-tree
    private int HT;                // height of the B-tree
    private int N;                 // number of key-value pairs in the B-tree

    // helper B-tree node data type
    private static final class Node {
        private int m;                                  // number of children
        private Entry[] children = new Entry[M];        // the array of children
        private Node(int k) { m = k; }                  // create a node with k children
    }

    // internal nodes: only use key and next
    // external nodes: only use key and value
    private static class Entry {
        private Comparable key;
        private Object value;
        private Node next;     // helper field to iterate over array entries
        public Entry(Comparable key, Object value, Node next) {
            this.key   = key;
            this.value = value;
            this.next  = next;
        }
    }

    // constructor
    public BTree() { root = new Node(0); }
```

Παρουσίαση Υλοποίησης

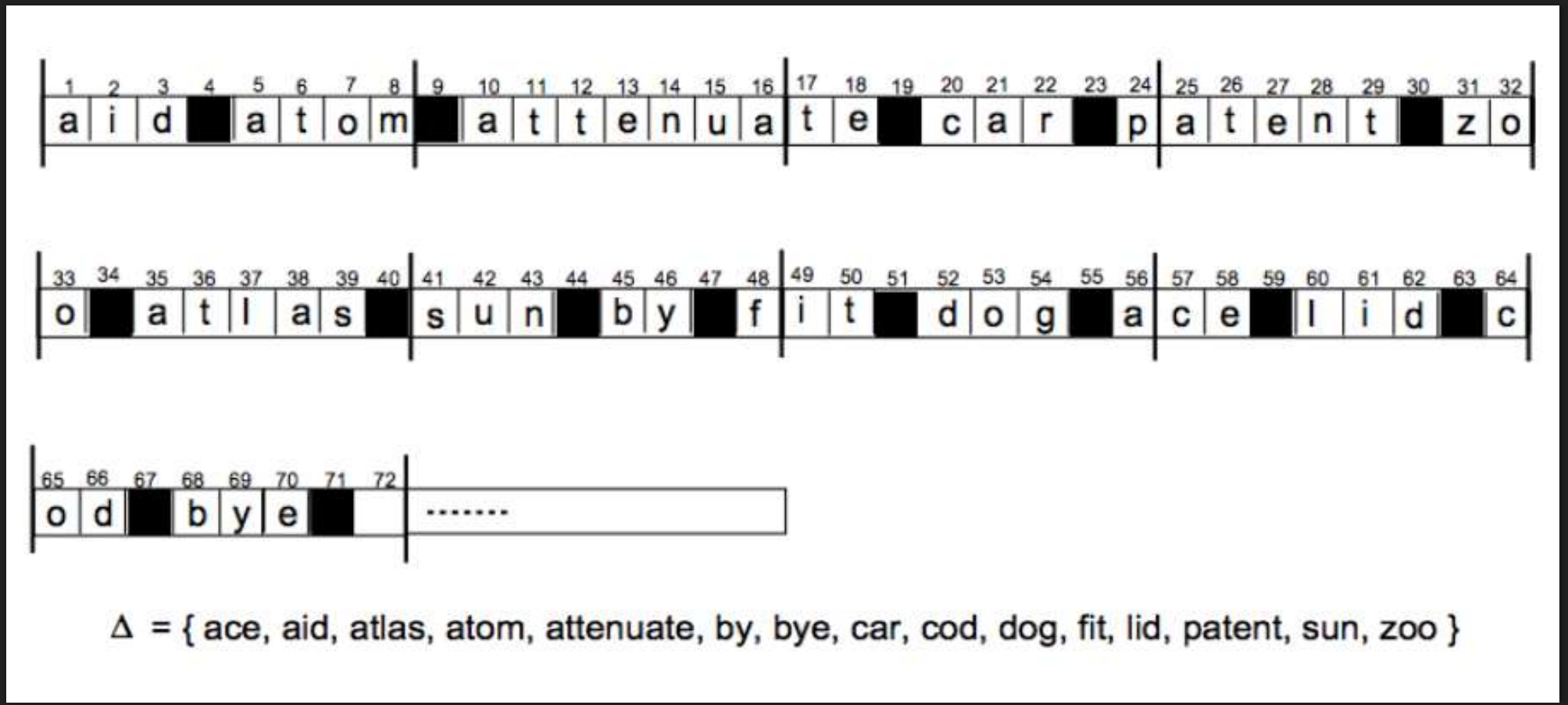
```
private Value search(Node x, Key key, int ht) {
    Entry[] children = x.children;

    // external node
    if (ht == 0) {
        for (int j = 0; j < x.m; j++) { // for all children of Node x
            // if the key is found, return the value
            if (eq(key, children[j].key)) return (Value) children[j].value;
        }
    }

    // internal node
    else {
        for (int j = 0; j < x.m; j++) { // for all children of Node x
            //if i'm inspecting the last child, or the key i'm searching is smaller than the next one..
            if (j+1 == x.m || less(key, children[j+1].key))
                // ... call search recursively
                return search(children[j].next, key, ht-1);
        }
    }
    return null;
}
```

Παρουσίαση Υλοποίησης

```
public static void main(String[] args) {
    BTree<String, String> st = new BTree<String, String>();

    st.put("1", "value1");
    st.put("2", "value2");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");

    st.put("3", "value3");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");
}
```



String B-tree



String B-tree