

B-Trees

Κ08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού

Κώστας Χατζηκοκολάκης

Searching on disks

- So far we have assumed that our data are stored in memory
- What about storing data on a **disk**?
 - **Example**: databases
- **Disk access** can be at least 100,000 to 1,000,000 slower
 - Goal: **minimize disk accesses**
- Also: data is read in **blocks**
 - Eg 512 or 1024 bytes
 - Reading 1 byte is the same as **reading a whole block**

B-Trees

We can easily generalize 2-3 / 2-4-trees:

- A **B-tree** of order m is an m -way search tree with 2 extra properties
- **Size property**
 - Any node contains between $\lceil \frac{m}{2} \rceil - 1$ **and** $m - 1$ **values**
 - **Internal** nodes with n values have exactly $n + 1$ **children**
 - The **root is excluded** from this rule
- **Depth property**
 - All **leaves** have the **same depth** (lie on the same level)

B-Trees

- We select m so that the **whole B-tree node** fits in a block
 - We read “multiple nodes” for the “price” of one
 - Fewer disk accesses than reading multiple nodes of a BST / AVL / ...
- Such trees are **balanced**
 - $h = O(\log n)$
 - Proof: exercise
- 2-3 and 2-4-trees are B-trees of order 3 and 4
- Generalize even more: require between a and b children
 - for some $2 \leq a \leq \frac{(b+1)}{2}$

ADT Set using BTree, types

```
// Κόμβος του set, περιέχει μία μόνο τιμή. Κάθε btree_node έχει πολλά
struct set_node {
    Pointer value;           // Η τιμή του κόμβου.
    BTreeNode owner;       // Ο btree_node στον οποίο ανήκει αυτό το
};

// Το struct btree_node είναι ο κόμβος ενός B-Δέντρου.
struct btree_node {
    int count;              // Αριθμός στοιχείων
    BTreeNode parent;      // Πατέρας
    BTreeNode children[MAX_CHILDREN + 1]; // Παιδιά
    SetNode set_nodes[MAX_VALUES + 1];    // Δεδομένα (μέσα σε set
};

// Υλοποιούμε τον ADT Set μέσω B-Tree, οπότε το struct set είναι ένα
struct set {
    BTreeNode root;        // Η ρίζα του δέντρου
    int size;              // Μέγεθος, για αποδοτικό set_size
    CompareFunc compare;  // Διάταξη
    DestroyFunc destroy_value; // Συνάρτηση που καταστρέφει ένα στοι
```

Insertion in a B-tree

- Same as for 2-3 and 2-3-4-trees
 - Search for the value
 - Insert at a leaf
- In case of an overflow ($m + 1$ children)
 - Split it into two nodes of $m/2$ children each
 - Move the separator value (median) to the parent

Insert example, $m = 5$

0

Insert example, $m = 5$

0	1
---	---

Inserting 1

Insert example, $m = 5$

0	1	2
---	---	---

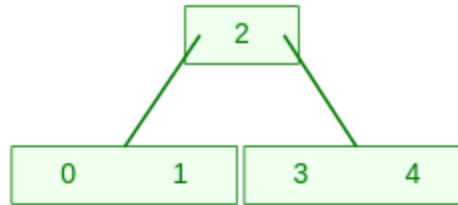
Inserting 2

Insert example, $m = 5$

0	1	2	3
---	---	---	---

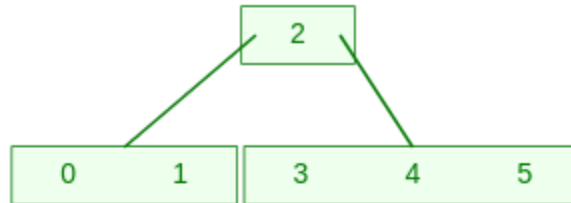
Inserting 3

Insert example, $m = 5$



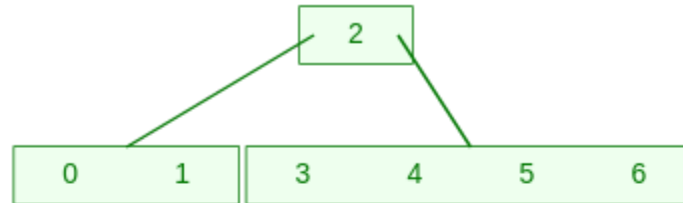
Inserting 4: overflow, 2 moves to a new root

Insert example, $m = 5$



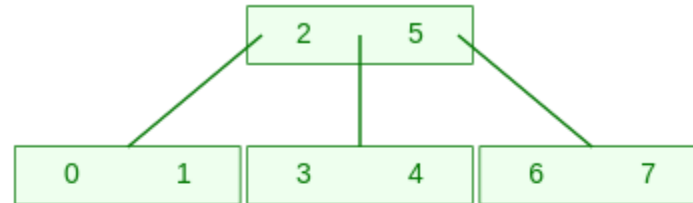
Inserting 5

Insert example, $m = 5$



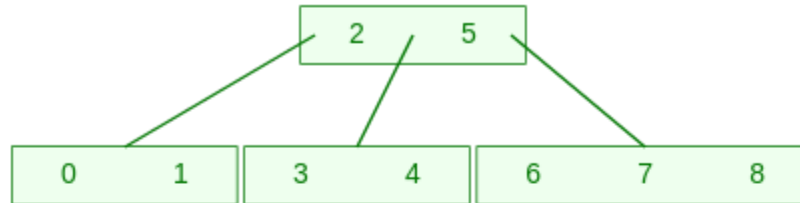
Inserting 6

Insert example, $m = 5$



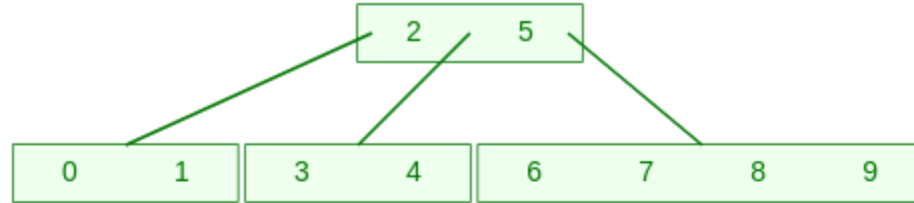
Inserting 7: overflow, 5 moves up

Insert example, $m = 5$



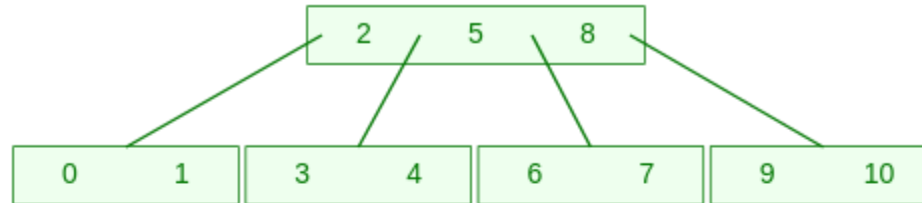
Inserting 8

Insert example, $m = 5$



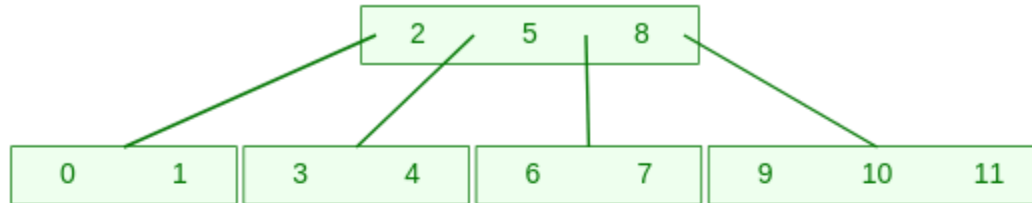
Inserting 9

Insert example, $m = 5$



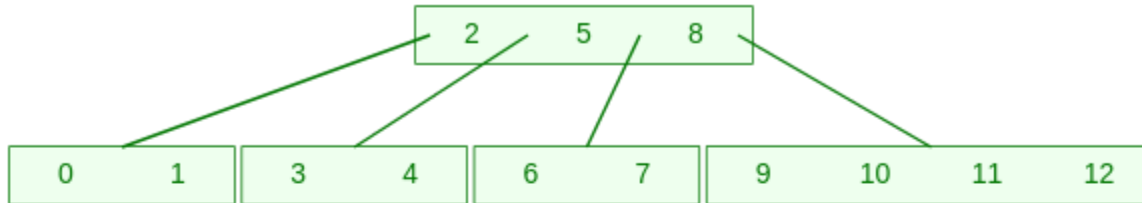
Inserting 10: overflow, 8 moves up

Insert example, $m = 5$



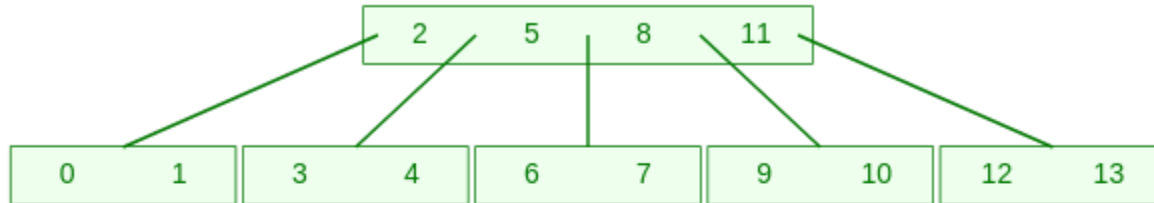
Inserting 11

Insert example, $m = 5$



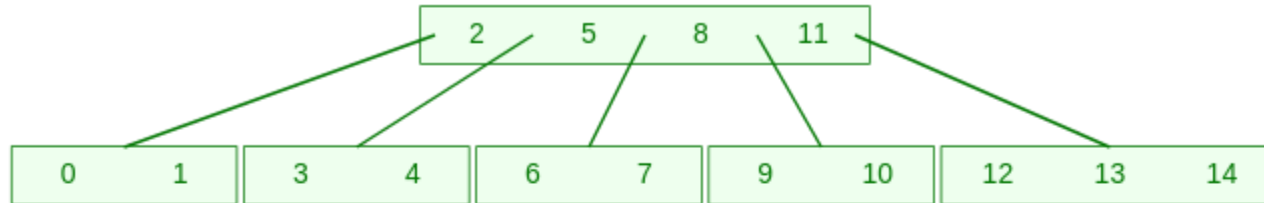
Inserting 12

Insert example, $m = 5$



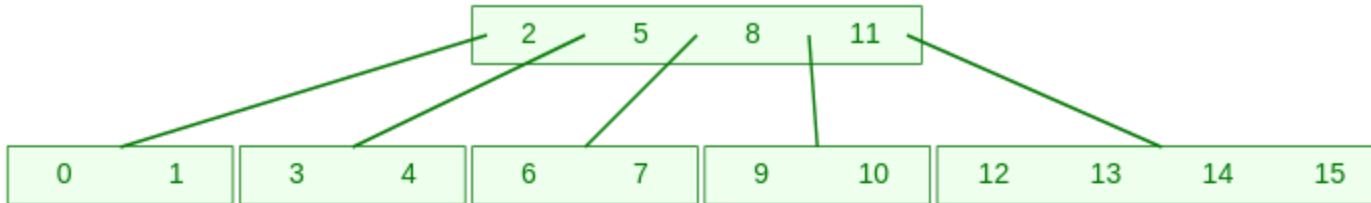
Inserting 13: overflow, 11 moves up

Insert example, $m = 5$



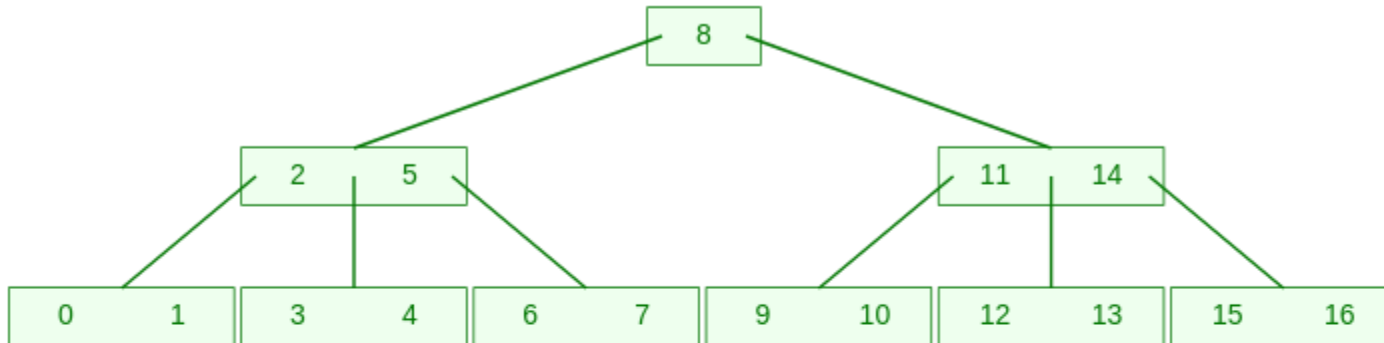
Inserting 14

Insert example, $m = 5$



Inserting 15

Insert example, $m = 5$



Inserting 16: overflow, 14 moves up, creating a new overflow

Code

- As always, the code is in `lecture-code`
 - `modules/UsingBTree/ADTSet.c`
- We only highlight some parts here

Code, node_insert

```
BTreeNode node_insert(BTreeNode root, CompareFunc compare, Pointer va
    // [απλός κώδικας για την περίπτωση κενού δέντρου]

    // Εύρεση του κόμβου στον οποίο πρέπει να γίνει insert
    int index;
    BTreeNode node = node_find(root, compare, value, &index);
    if (index != -1) { // Υπάρχει ήδη η τιμή
        node->set_nodes[index]->value = value;
        return root;
    }

    // Εύρεση της θέσης που πρέπει να μπει το value
    for (index = 0;
        index < node->count && compare(value, node->set_nodes[index]
            index++)
        ;

    node_add_value(node, set_node_create(value), index);

    if (node->count > MAX_VALUES) // overflow
        split(node, compare);

    // Επιστρέφουμε τη ρίζα, μπορεί να έχει δημιουργηθεί νέα
    return root->parent != NULL ? root->parent : root;
}
```

Code, split

```
// Καλείται όταν ο κόμβος node έχει υπερχειλήσει, τον χωρίζει σε 2 κό  
// Στέλνει τη μεσαία από τις τιμές του κόμβου node στον πατέρα του.
```

```
static void split(BTreeNode node, CompareFunc compare) {  
    // Χωρίζουμε τον κόμβο node σε 2 κόμβους  
    BTreeNode right = node_create();  
    right->parent = node->parent;    // Οι 2 κόμβοι έχουν τον ίδιο π  
  
    // Μετακίνησε τις μισές τιμές και παιδιά από τον αριστερό κόμβο ο  
    int half = node->count/2;  
    if (!is_leaf(node))  
        for (int i = 0; i <= half; i++)  
            node_add_child(right, node->children[i + half + 1], i);  
  
    for (int i = 0; i < half; i++) {  
        node_add_value(right, node->set_nodes[i + half + 1], i);  
        node->count--;  
    }  
  
    // Αφαίρεση μεσαίας τιμής  
    SetNode median = node->set_nodes[node->count-1];  
    node->count--;  
  
    ...
```

Code, split

```
// Προσθέτουμε το median στον πατέρα του κόμβου node.
BTreeNode parent = node->parent;
if (parent == NULL) { // 0 node είναι η ρίζ
    BTreeNode new_root = node_create(); // Δημιούργησε καινού

    node_add_value(new_root, median, 0);

    right->parent = node->parent = new_root;
    new_root->children[0] = node;
    new_root->children[1] = right;

} else {
    int index; // Βρες τη θέση εισαγωγής της τιμής στον πατέρα.
    for (index = 0; index < parent->count; index++)
        if (compare(median->value, parent->set_nodes[index]->valu
            break);

    // Πρόσθεσε τον right ως δεξιό παιδί της (νέας) διαχωριστικής
    node_add_child(parent, right, index+1);
    node_add_value(parent, median, index);

    if (parent->count > MAX_VALUES) // parent overflows
        split(parent, compare);
}
}
```

Removal from B-trees

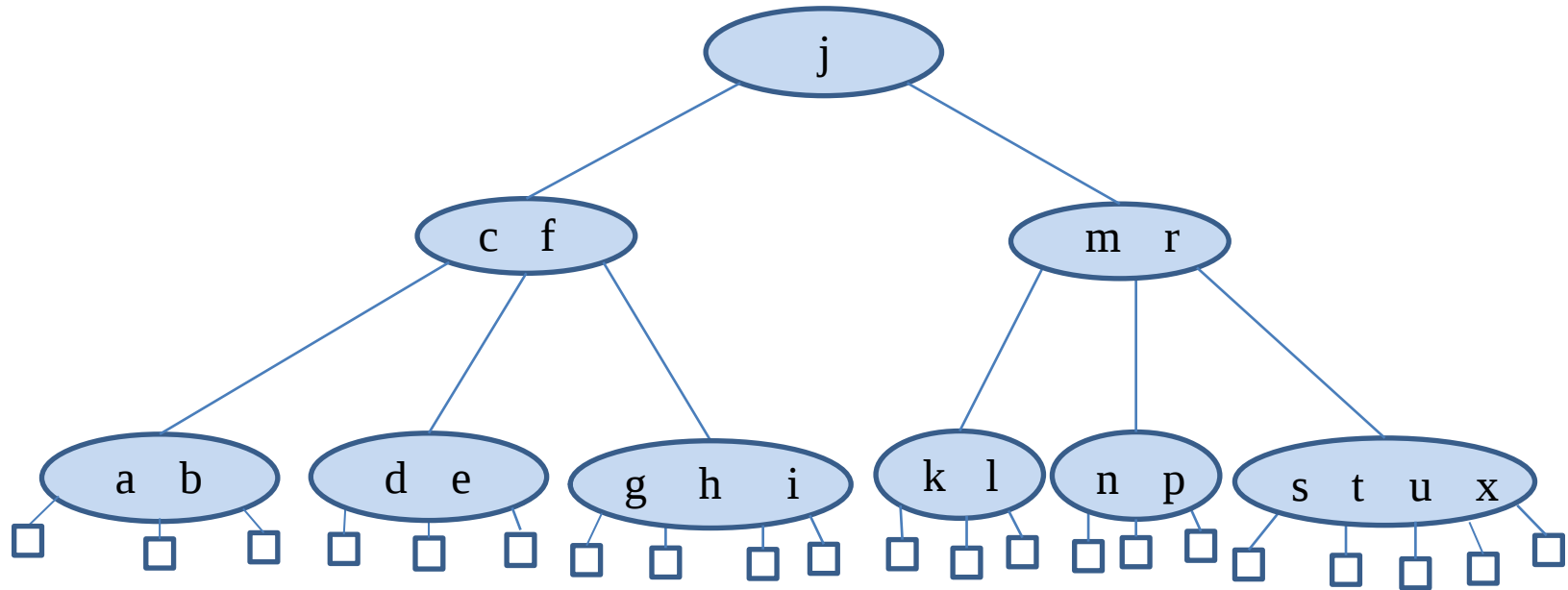
- Same as for 2-3 and 2-3-4-trees
- To remove a value k_i from an **internal** node
 - Replace with its **predecessor** (or its **successor**)
 - Right-most value in the i -th subtree
- To remove a value from a **leaf**
 - We simply remove it
 - But it might violate the **size** property (**underflow**)

Fixing underflows

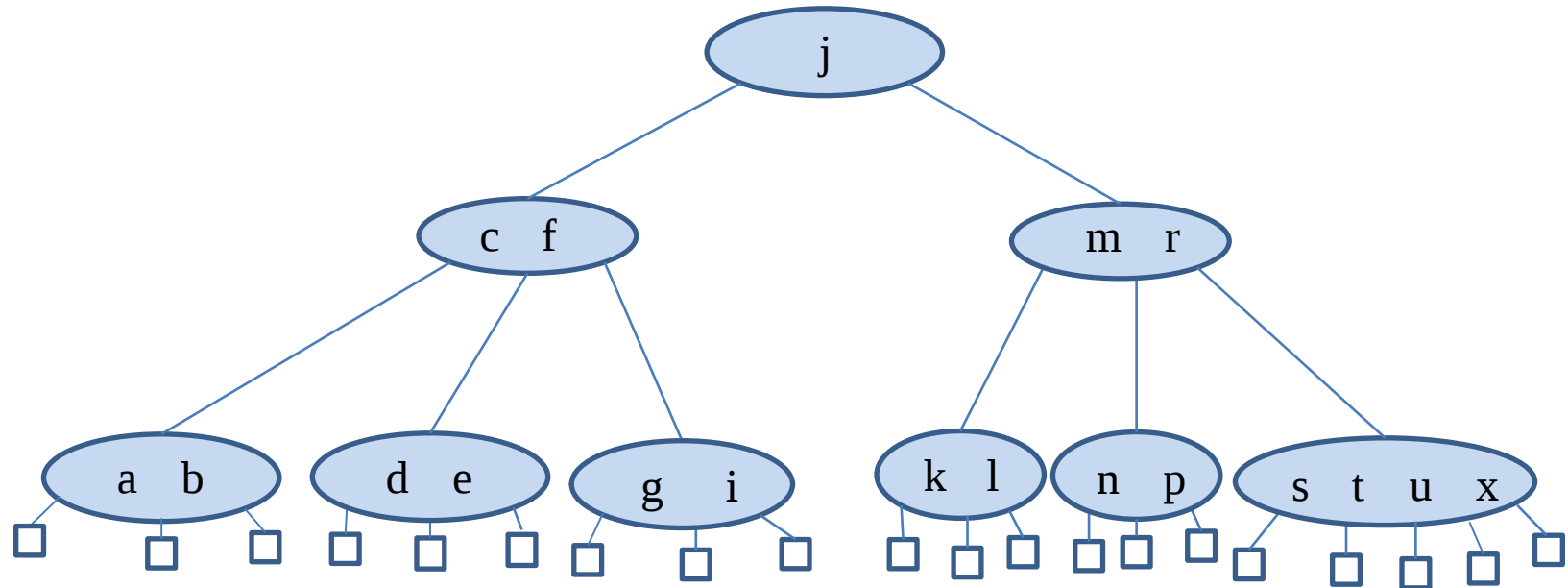
Two strategies for fixing an underflow at ν

- Is there an **immediate sibling** w with a “spare” value?
- If so, we do a **transfer** operation
 - Move a value of w to its parent u
 - Move a value of the parent u to ν
- If not, we do a **fusion** operation
 - Merge ν and w , creating a new node ν'
 - Move a value from the parent u to ν'
 - This might **underflow the parent**, continue the same procedure there

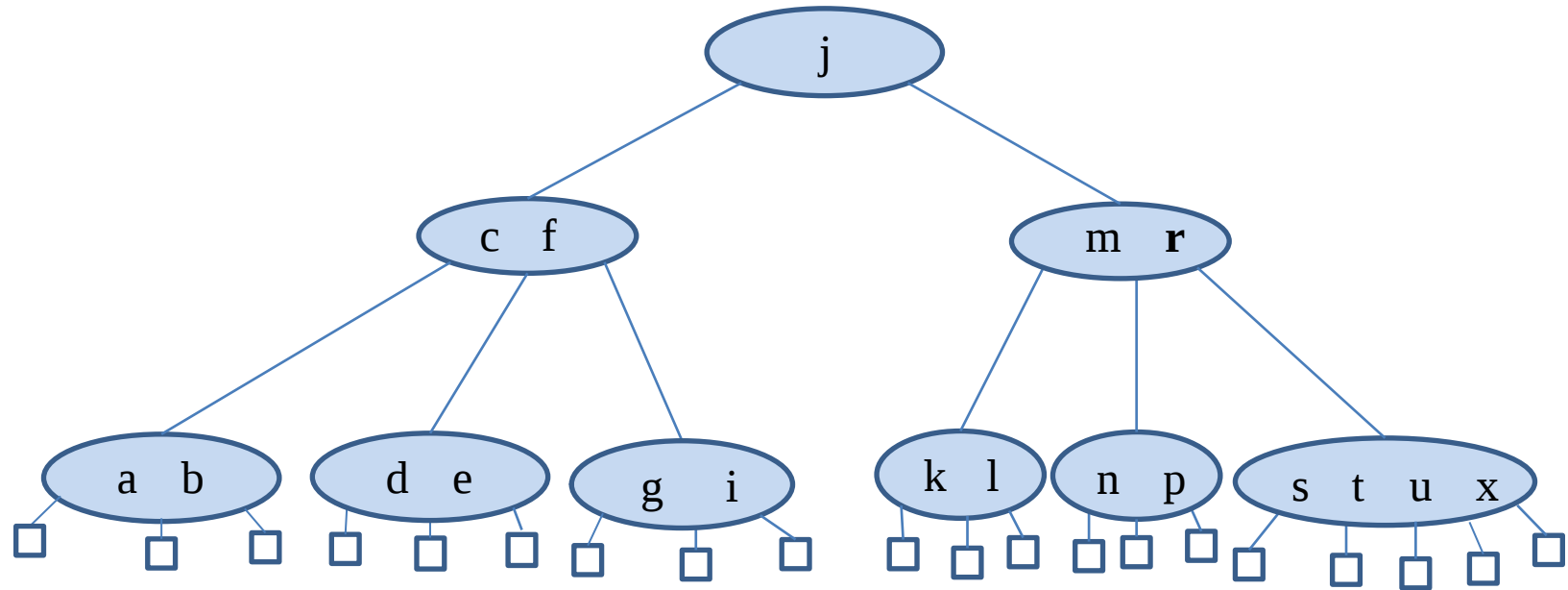
Example



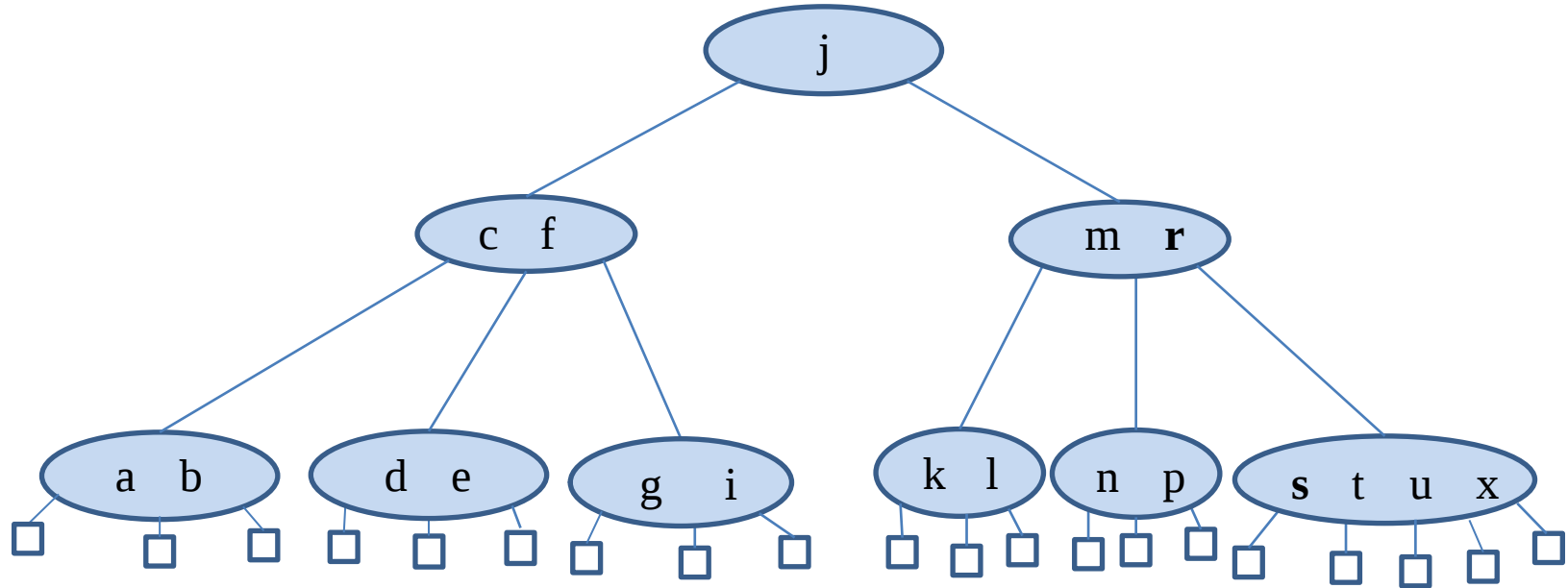
Delete h



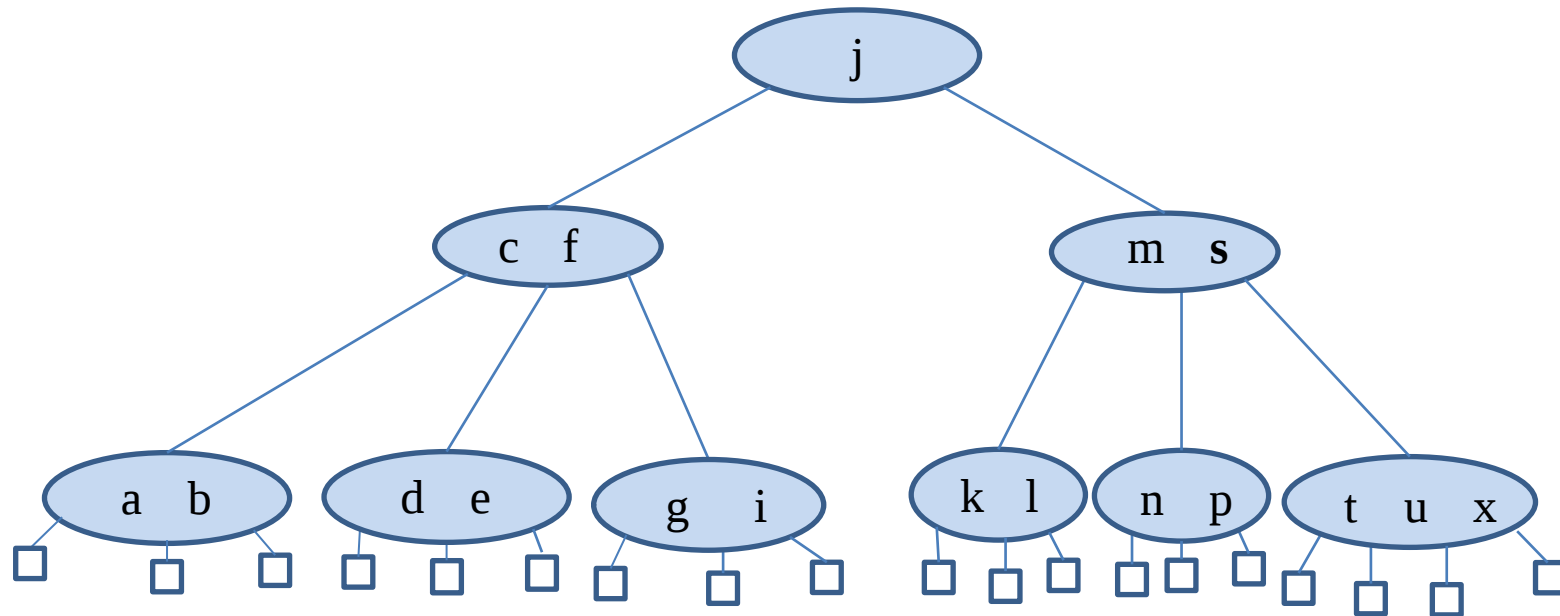
Delete r



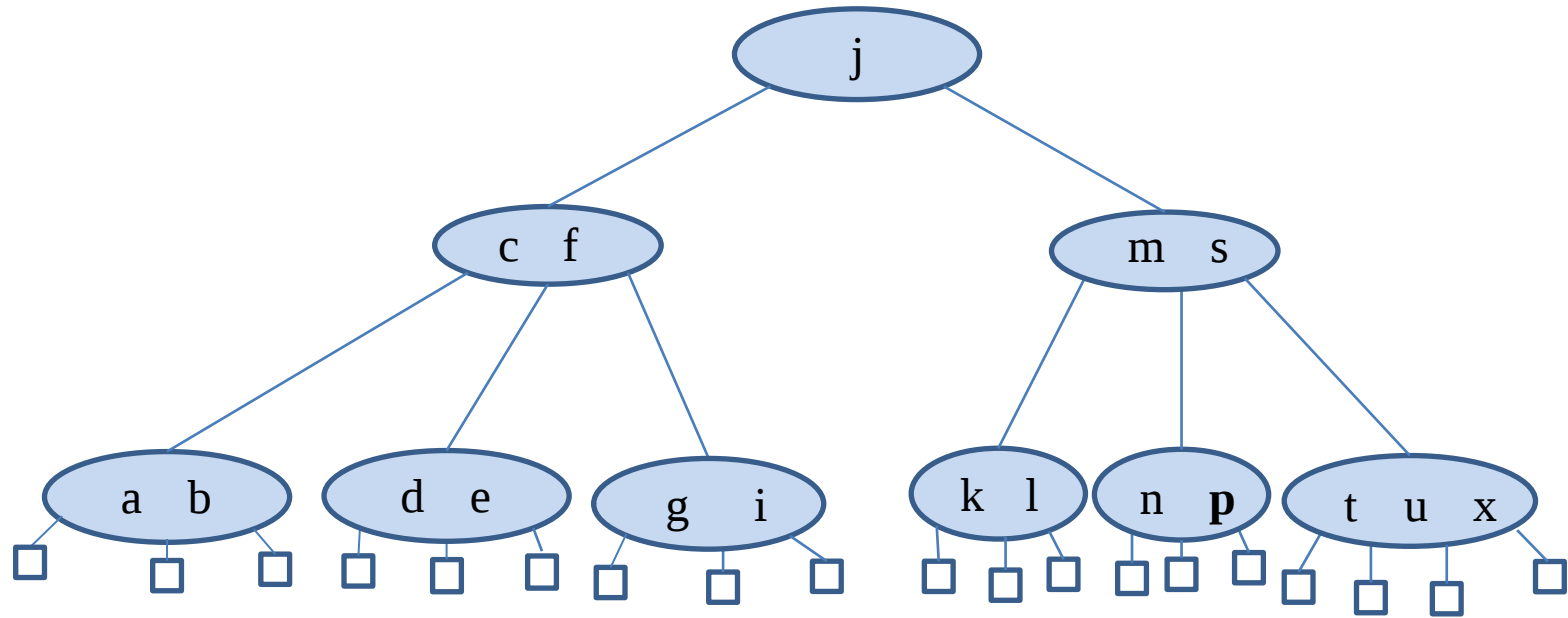
Find the Successor of r



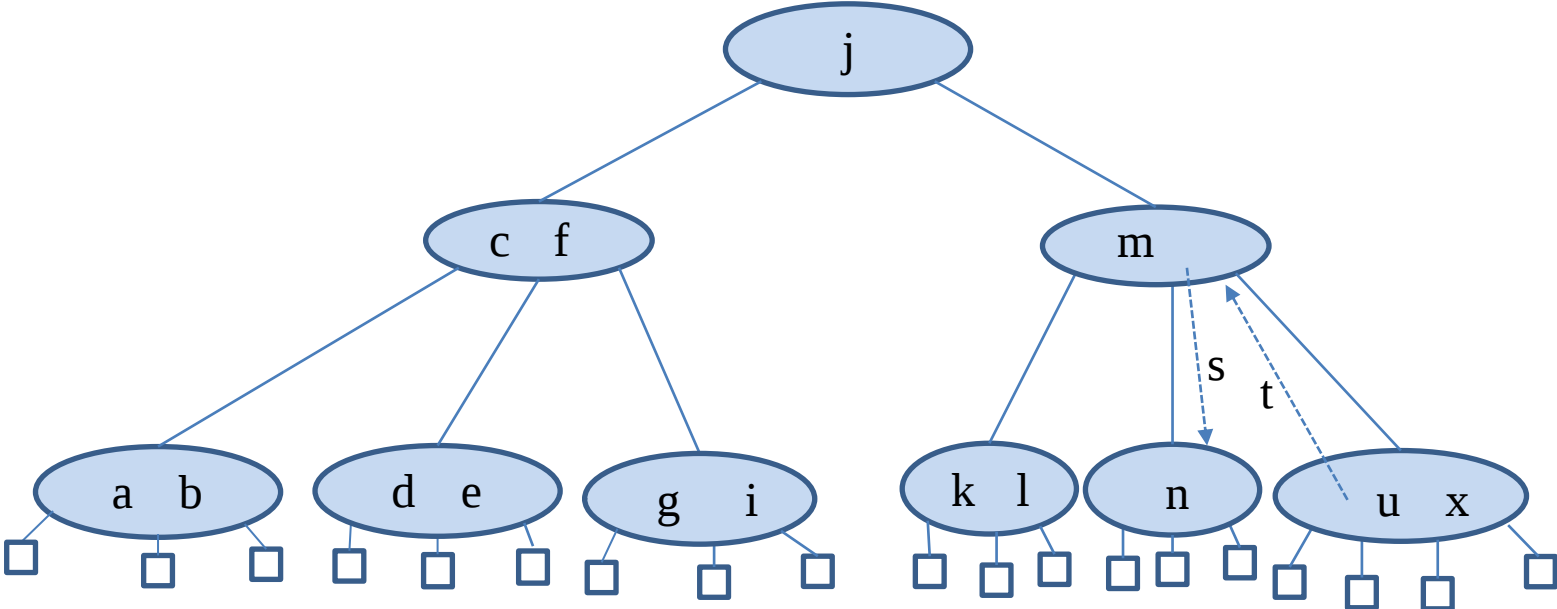
Promote the Successor of r – Delete the Successor from its Place



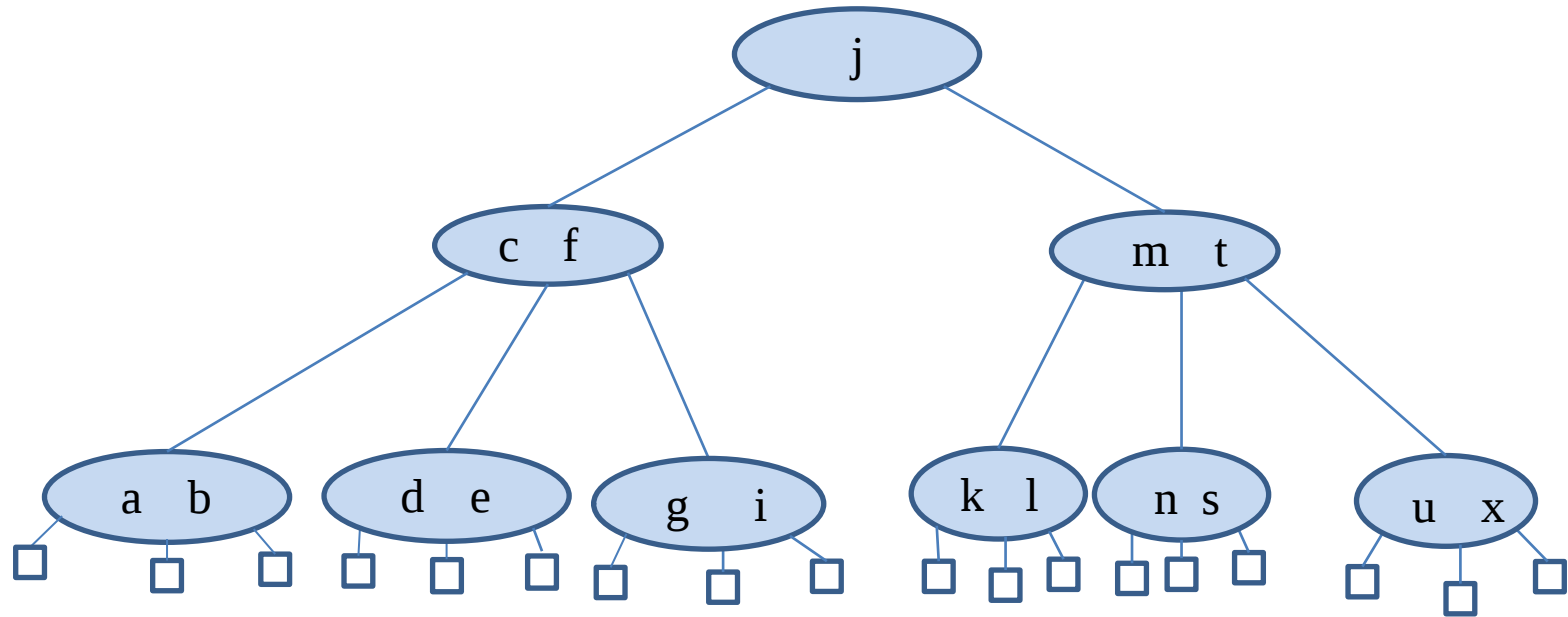
Delete p



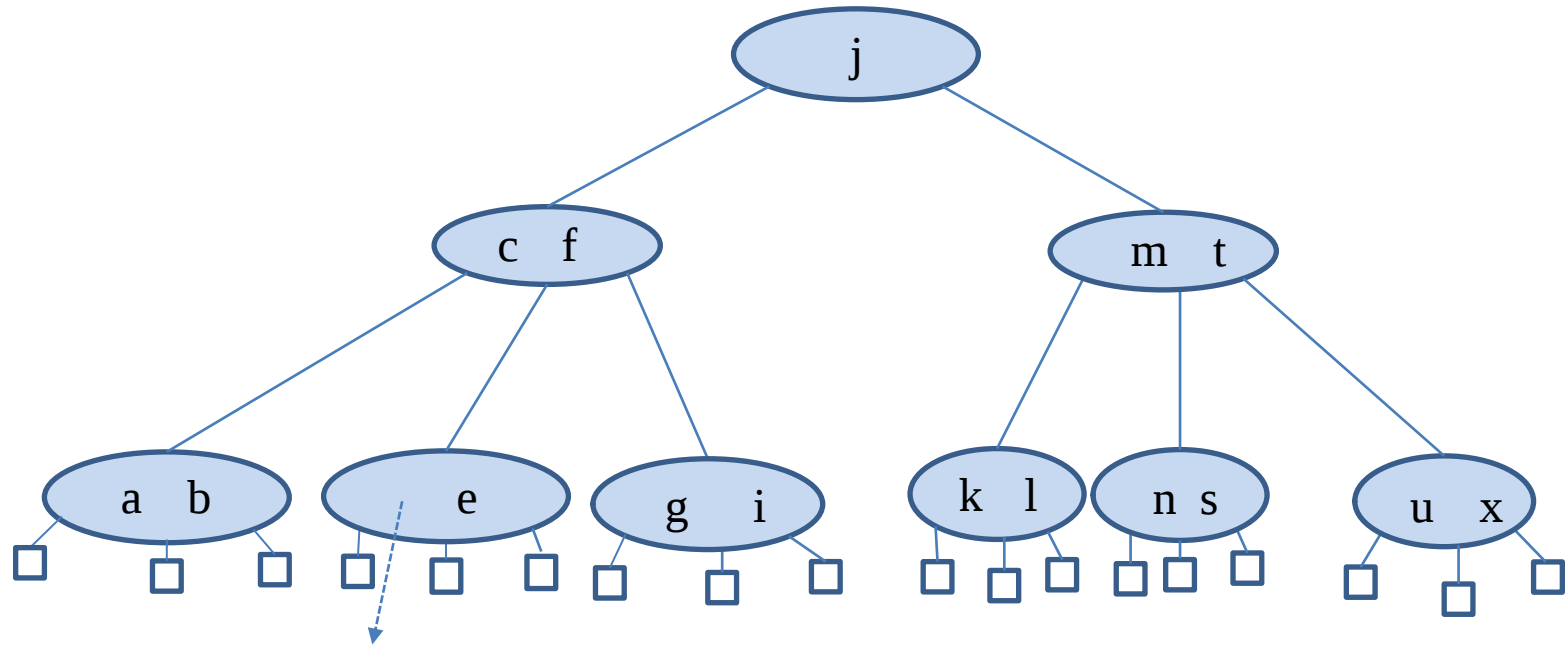
Transfer



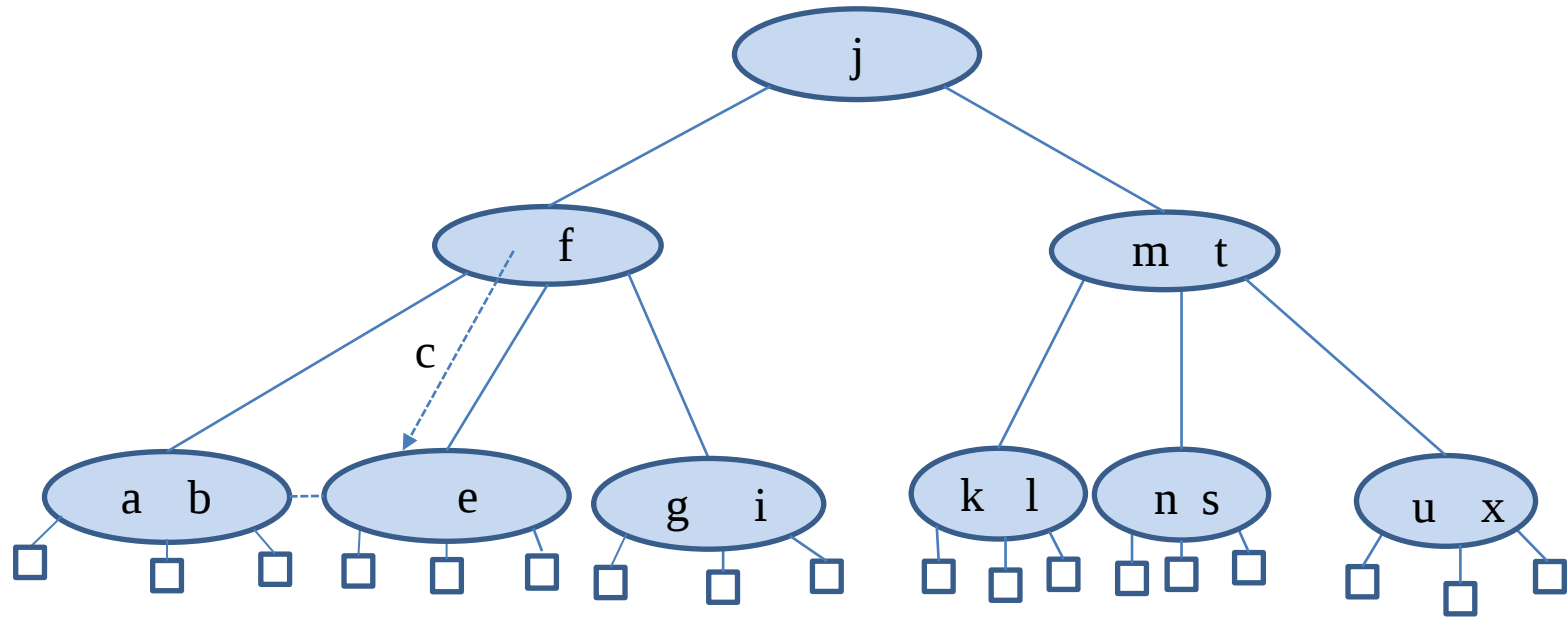
After the Transfer



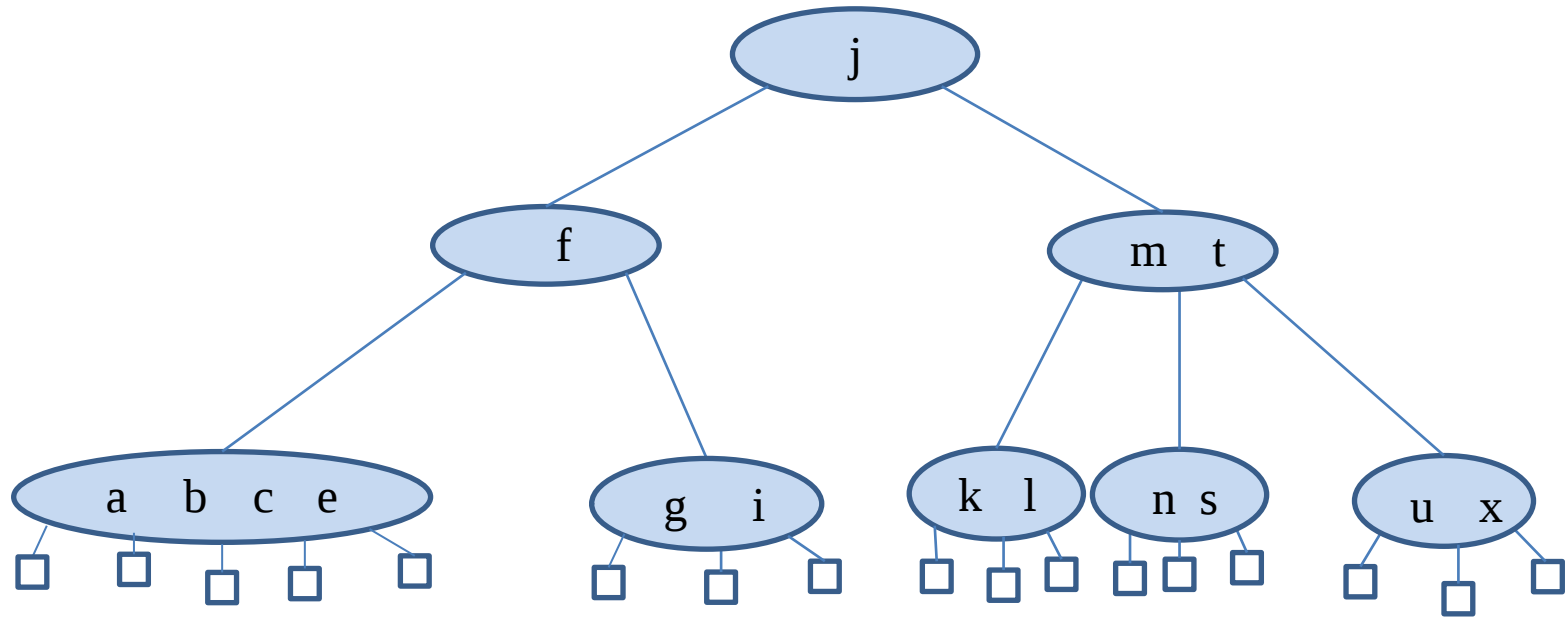
Delete d



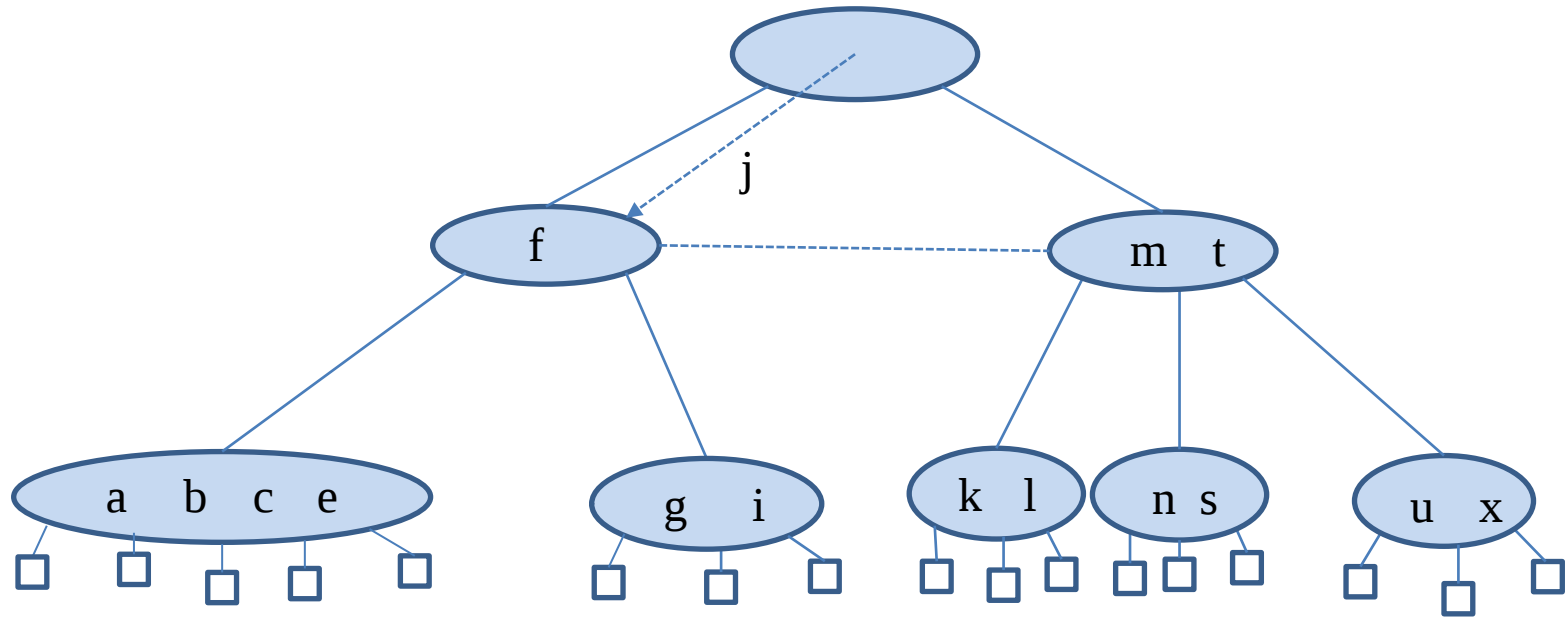
Fusion



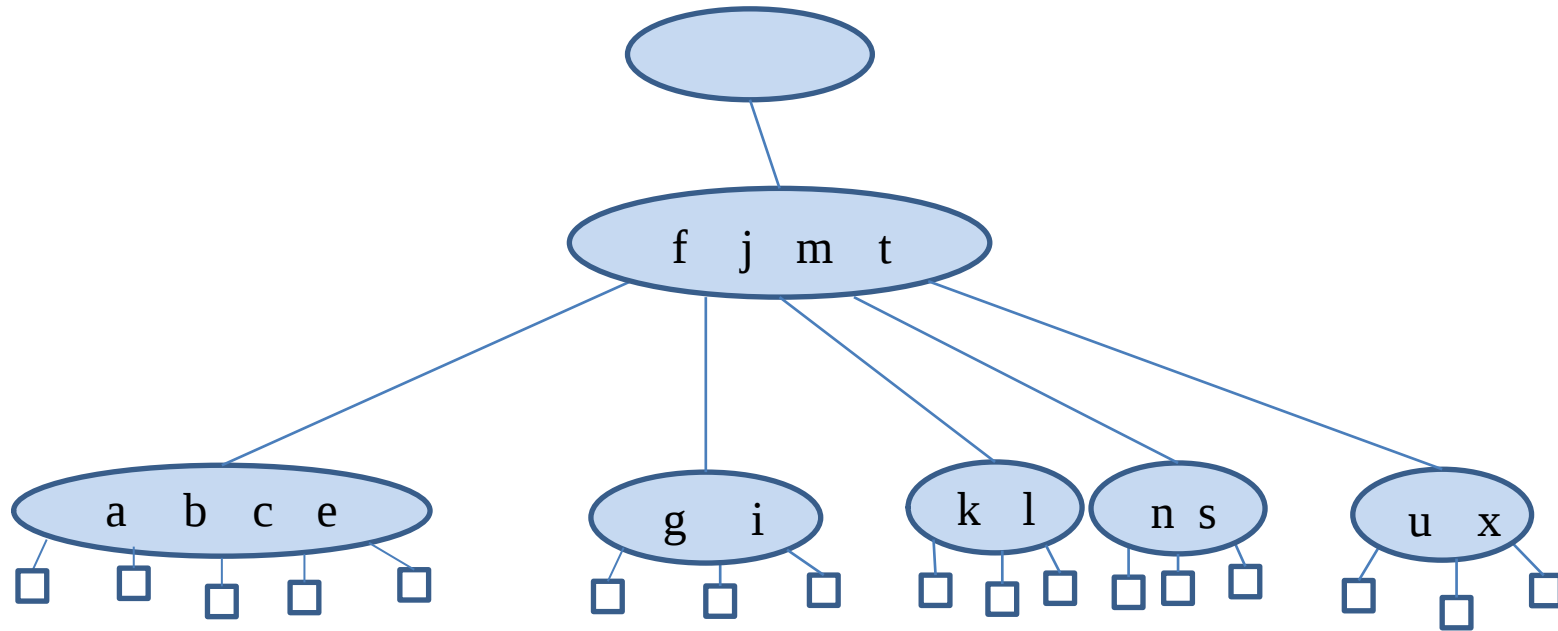
After the Fusion – Underflow at f



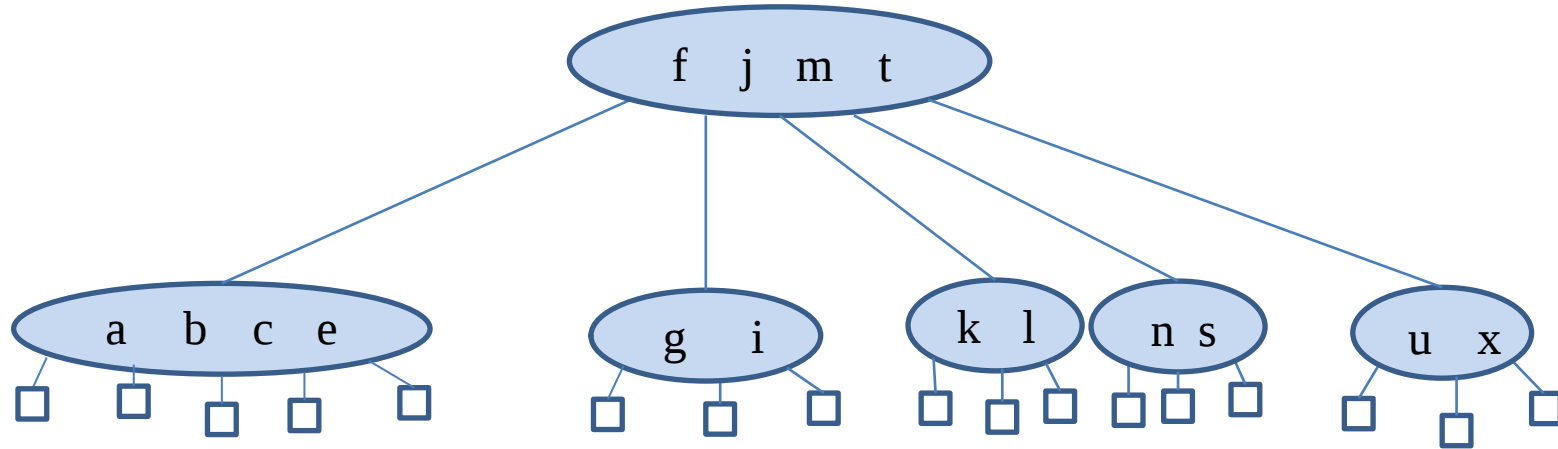
Fusion



After the Fusion – Delete Root



Final Tree



Code, node_remove

```
BTreeNode node_remove(BTreeNode root, CompareFunc compare, Pointer va
    // Βρες τον κόμβο που περιέχει την τιμή.
    int index;
    BTreeNode node = node_find(root, compare, value, &index);

    if (index == -1)    // Η τιμή δεν υπάρχει στο δέντρο.
        return root;

    // Βρέθηκε ισοδύναμη τιμή στον node, οπότε τον διαγράφουμε
    // Το πώς θα γίνει αυτό εξαρτάται από το αν έχει παιδιά.

    if (is_leaf(node)) {
        // Φύλλο: διάγραψε την τιμή, αναδιάταξε τα δεδομένα, repair
        // Ολίσθησε όλα τα δεδομένα 1 θέση αριστερά.
        for (int i = index; i < node->count-1; i++)
            node->set_nodes[i] = node->set_nodes[i + 1];

        node->count--;    // Αφαίρεσε το δεδομένο.

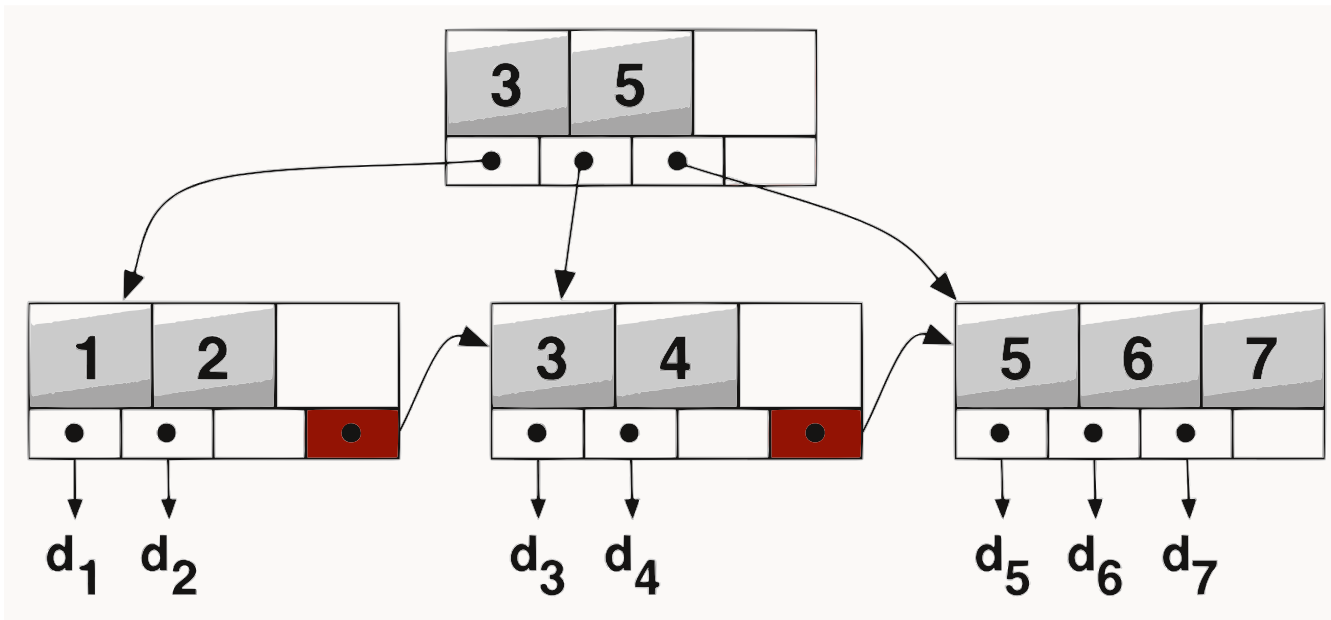
        repair_underflow(node);    // Αναδιαμόρφωσε το δένδρο.
```

Code, node_remove

```
} else {  
    // Αν είναι εσωτερικός κόμβος αντικατάσταση με την next τιμή  
    // και remove της τιμής αυτής  
    SetNode max = node_find_max(node->children[index]);  
  
    BTreeNode max_node = max->owner;  
    max_node->count--;    // Αφαίρεσε το δεδομένο.  
  
    node->set_nodes[index] = max;  
    max->owner = node;  
  
    repair_underflow(max_node);    // Αναδιαμόρφωσε το δέντρο.  
}  
  
// Αν η ρίζα αδειάσει, ρίζα γίνεται το (μοναδικό, αν έχει) παιδί  
if (root->count == 0) {  
    BTreeNode first_child = root->children[0];  
    if (first_child != NULL)  
        first_child->parent = NULL;  
    root = first_child;  
}  
return root;  
}
```

B+-trees

A variant of B-trees, important in today's **file systems** and **databases**.



Readings

- M. T. Goodrich, R. Tamassia and D. Mount. Data Structures and Algorithms in C++. 2nd edition. John Wiley.
- Sartaj Sahni. Δομές Δεδομένων, Αλγόριθμοι και Εφαρμογές στη C++. Εκδόσεις Τζιόλα.
- R. Sedgwick. Αλγόριθμοι σε C. Κεφ. 16.3