

Multi-Way Search Trees

Κ08 Δομές Δεδομένων και Τεχνικές Προγραμματισμού

Κώστας Χατζηκοκολάκης

1

Motivation

- We keep the **ordering** idea of BSTs
 - **Fast search**, by excluding whole subtrees
- And add **more than two children** for each node
 - Gives more flexibility in restructuring the tree
 - And new ways to **keep it balanced**

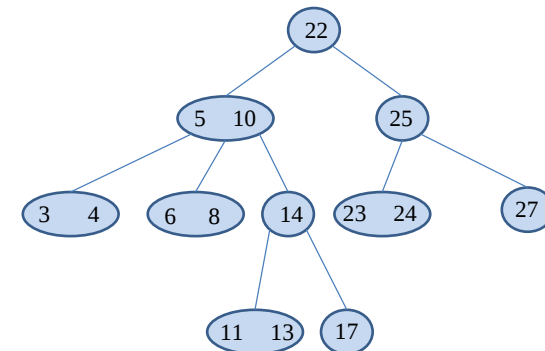
2

Multi-way search trees

- d -node: a node with d children
- Each **internal** d -node stores $d - 1$ **ordered** values $k_1 < \dots < k_{d-1}$
 - **No duplicate** values in the whole tree
- All values in a **subtree** lie **in-between** the corresponding node values
 - For all values l in the i -th subtree: $k_{i-1} < l < k_i$
 - Convention: $k_0 = -\infty, k_d = +\infty$
- m -way search tree: all nodes have **at most** m children
 - A BST is a 2-way search tree

3

Example multi-way search tree



$m = 3$

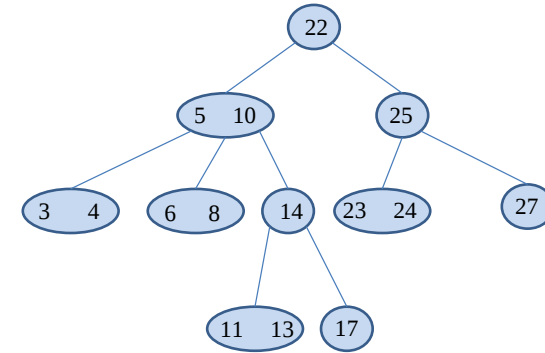
4

Searching in a multi-way search tree

- Simple adaptation of the algorithm for BSTs
- Start from the root, traverse towards the leaves
- In each node, there is **a single subtree** that can possibly contain a value l
 - The subtree i such that $k_{i-1} < l < k_i$
 - Continue in that subtree

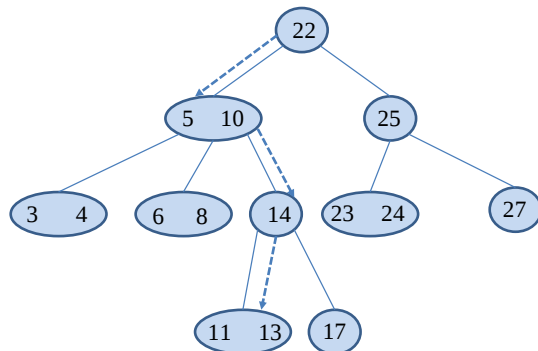
5

Example multi-way search tree



6

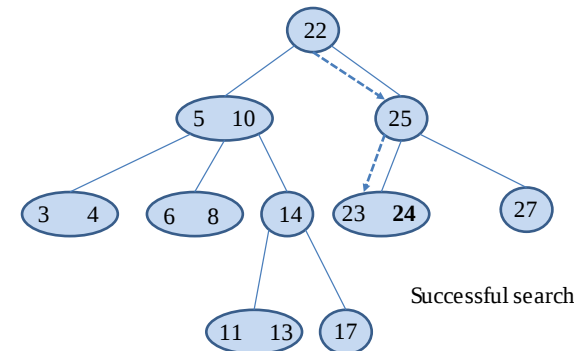
Search for value 12



Unsuccessful search

7

Search for value 24



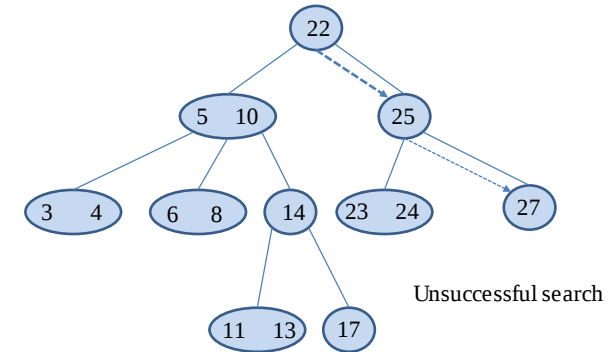
8

Insertion in a multi-way search tree

- Again, simple adaptation of BSTs
 - **But:** we don't always need to create a new node
 - We can insert in an existing one if there is space
- Start with a search for the value l we want to insert
- If found, stop (no duplicates)
- If not found, insert at the **leaf** we reached
 - If full, create an i -th child, such that $k_{i-1} < l < k_i$

9

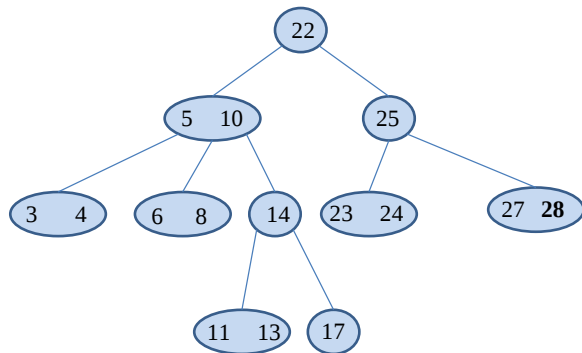
Insert value 28



$m = 3$

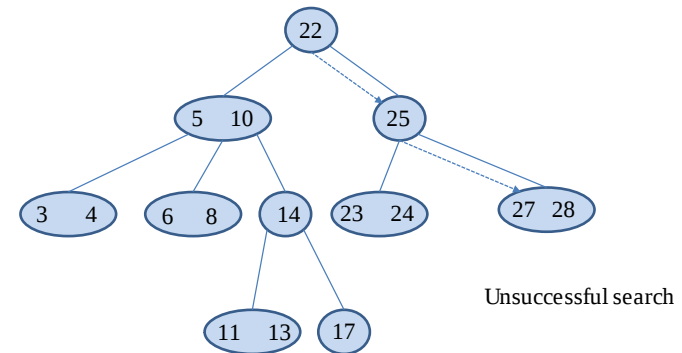
10

Value 28 inserted



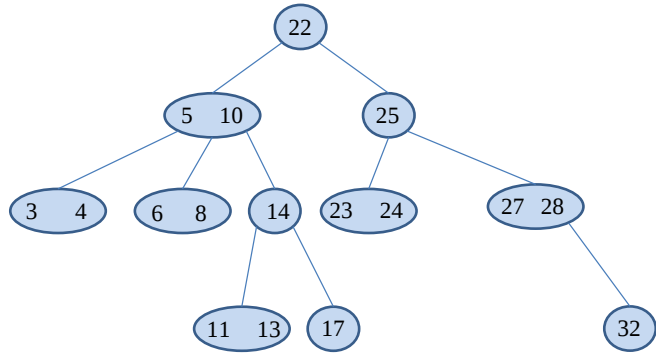
11

Insert value 32



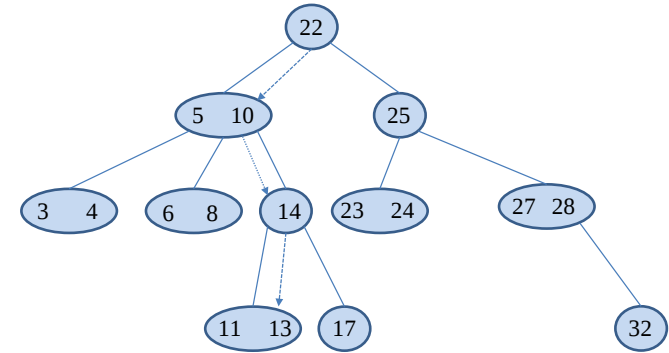
12

Value 32 inserted



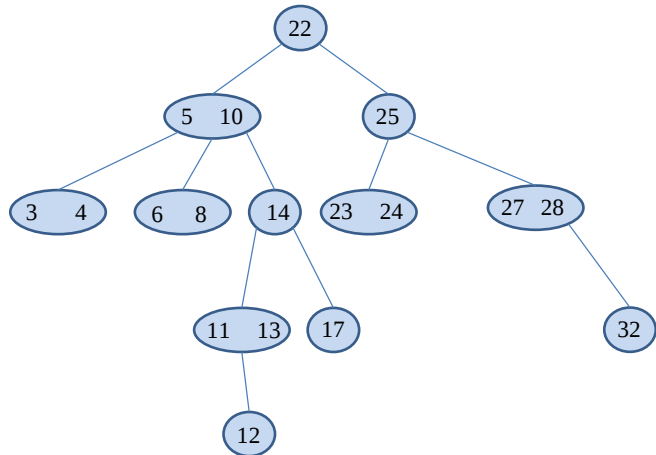
13

Insert value 12



14

Value 12 inserted



15

Deletion from a multi-way search tree

Left as an exercise.

16

Complexity of operations

- We need to traverse the tree from the root to a leaf
- The time spent at each node is constant
 - Eg. find i such that $k_{i-1} < l < k_i$
 - Assuming m is **fixed**!
- So as usual all complexities are $O(h)$
 - $O(n)$ in the worst-case

17

Balanced multi-way search trees

- Similarly to BSTs we need to keep the tree **balanced**
 - So that $h = O(\log n)$
- AVL where a kind of balanced BSTs
- We will study two kinds of **balanced multi-way** search trees:
 - **2-3 trees**
 - **2-3-4 trees** (also known as 2-4 trees)

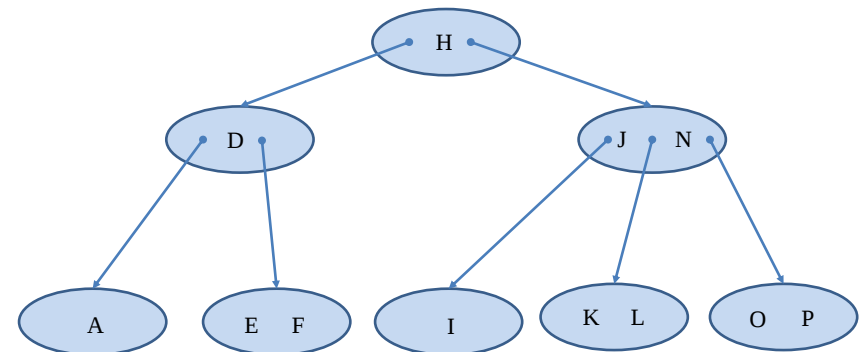
18

2-3 trees

- A **2-3 tree** is a 3-way search tree which has the following properties
- **Size property**
 - Each node contains **1 or 2 values**
 - **Internal** nodes with n values have exactly $n + 1$ **children**
- **Depth property**
 - All **leaves** have the **same depth** (lie on the same level)

19

Example of 2-3 tree



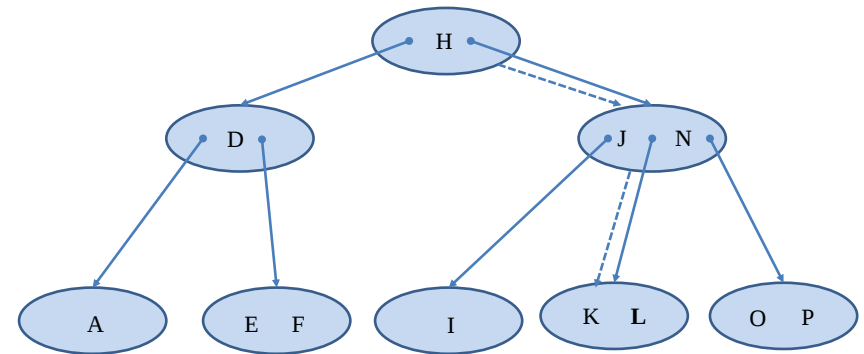
20

Height of 2-3 trees

- **All nodes** at **all levels** except the last one are **internal**
 - And each internal node has at least 2 children
 - So at level i we have at least 2^i nodes
- Hence $n \geq 2^h$, in other words $h = O(\log n)$
- So we can search for an element in time $O(\log n)$
 - Using the standard algorithm for m -way trees

21

Search for L



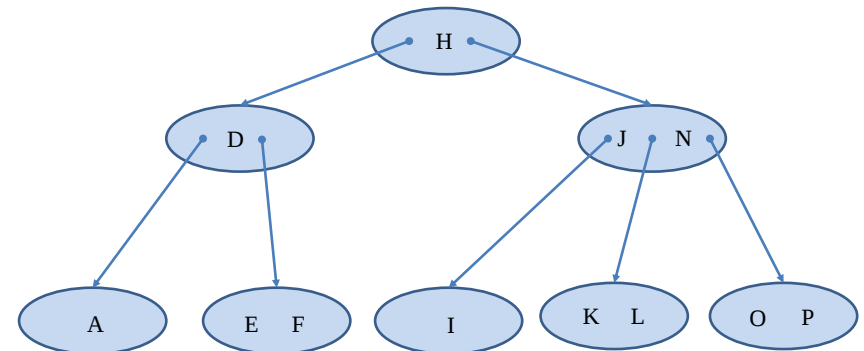
22

Insertion in 2-3-trees

- We can start by following the generic algorithm for m -way trees
- Search for the value l we want to insert
- If found, stop (no duplicates)
- If not found, insert at the **leaf** we reached

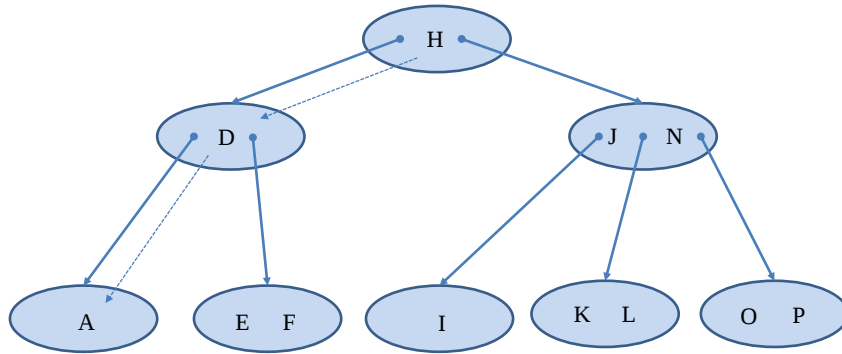
23

Example: insert B



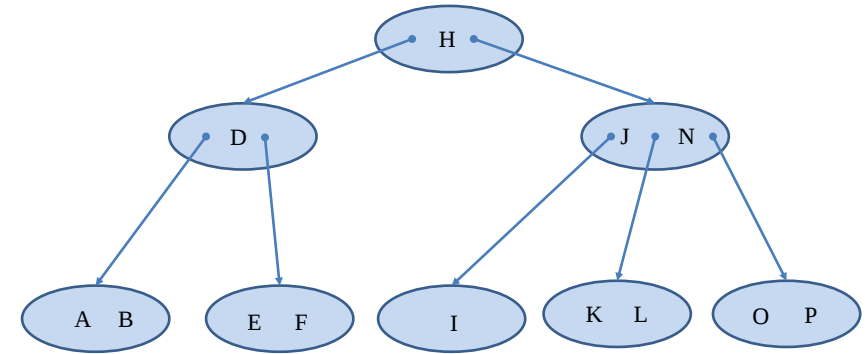
24

Example: insert B



25

Example: result



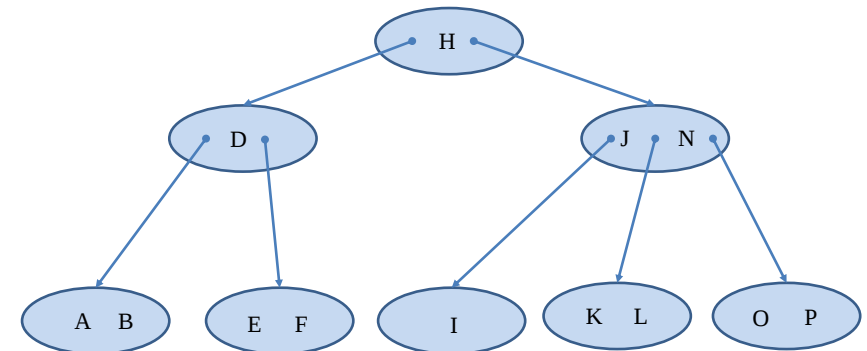
26

Insertion in 2-3-trees

- But what if there is **no space at the leaf** (overflow)?
- The standard algorithm will insert a child at the leaf
 - But this **violates the depth property!**
 - The new leaf is not at the same level
- Different strategy
 - **split** the overflowed node into two nodes
 - pass the **middle value** to the parent (**separator** of the two nodes)
- The middle value might **overflow the parent**
 - Same procedure: split and send the middle value up

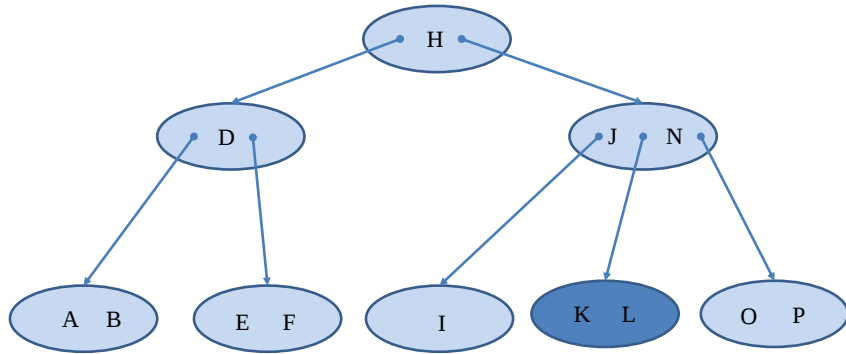
27

Example: insert M



28

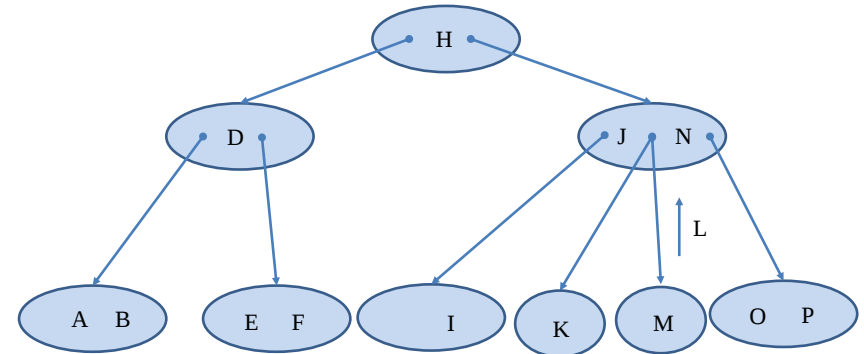
Example: insert M



M overflows this node.

29

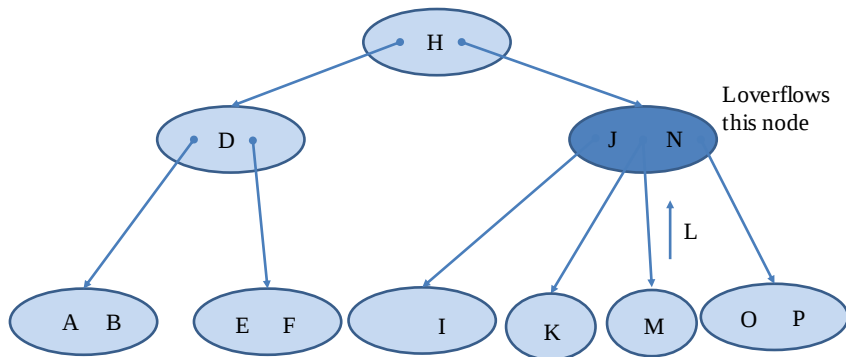
Example: insert M



The node is split in two and L is passed to the parent node

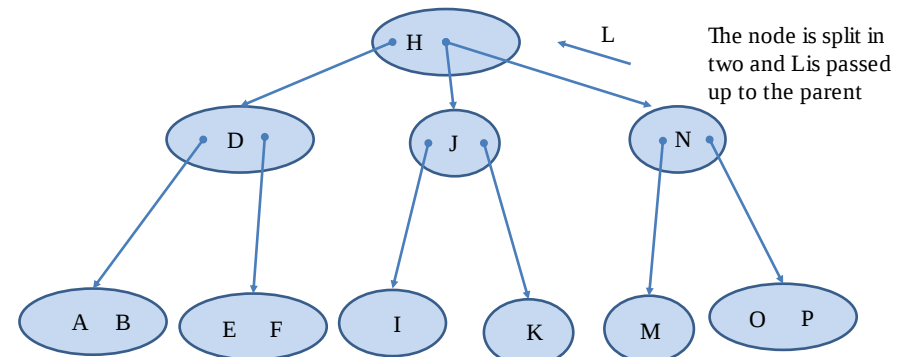
30

Example: insert M



31

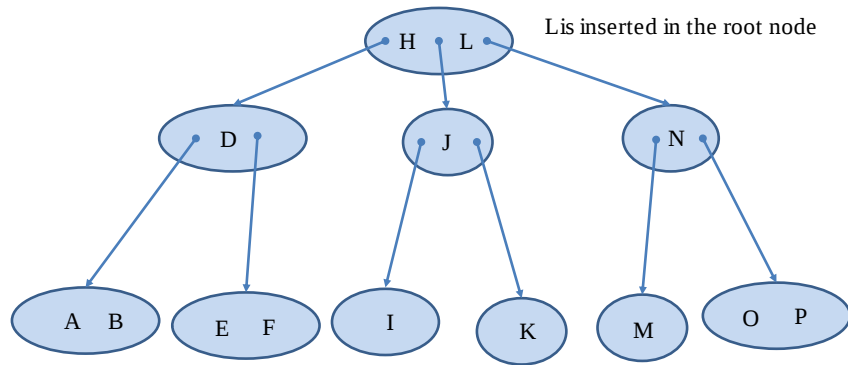
Example: insert M



The node is split in two and L is passed up to the parent

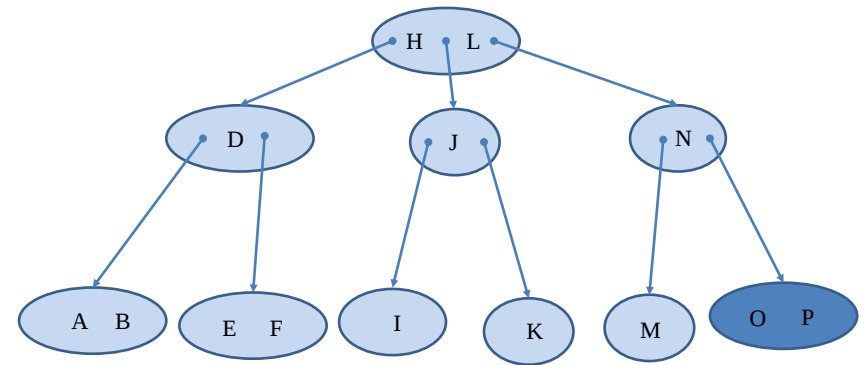
32

Example: result



33

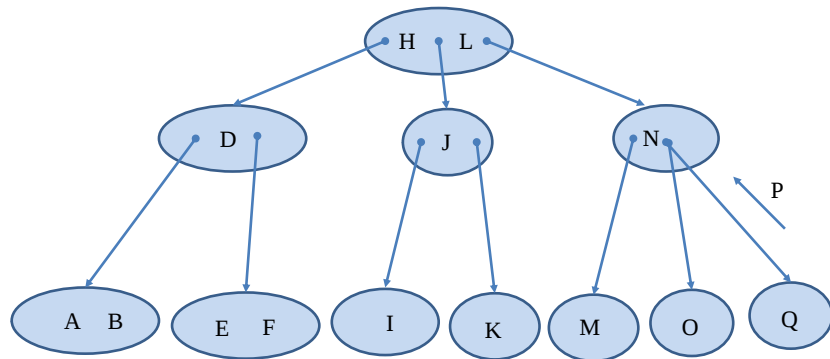
Example: insert Q



Q overflows
this node

34

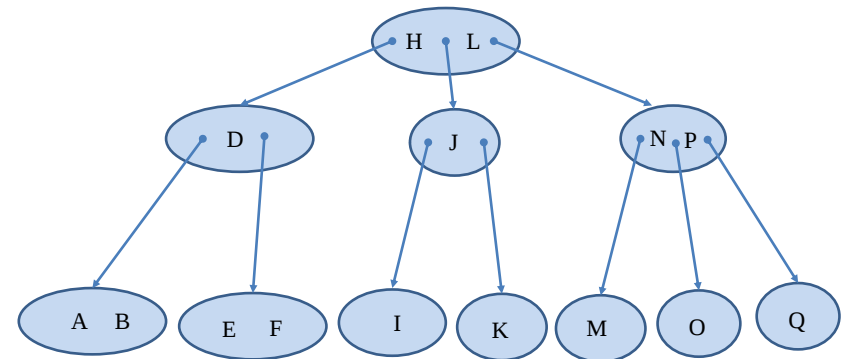
Example: insert Q



This node is split up
and P is passed up

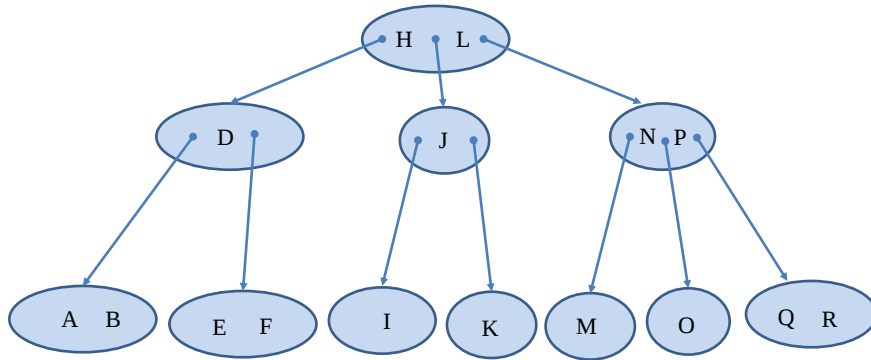
35

Example: result



36

Example: insert R



R is inserted in the node with Q where there is space.

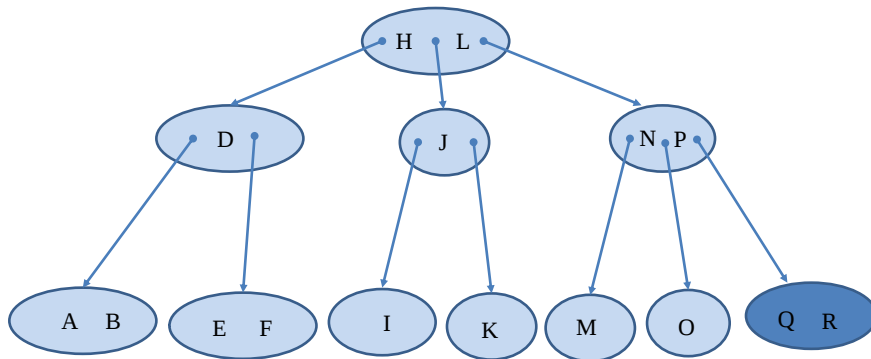
37

Insertion in 2-3-trees

- The **root** might also **overflow**
- Same procedure
 - Split it
 - The middle value moves up, creating a **new root**
- This is the **only** operation that **increases** the tree's **height**
 - It increases the depth of **all nodes** simultaneously
 - 2-3-trees grow at the root, not at the leaves!

38

Example: insert S

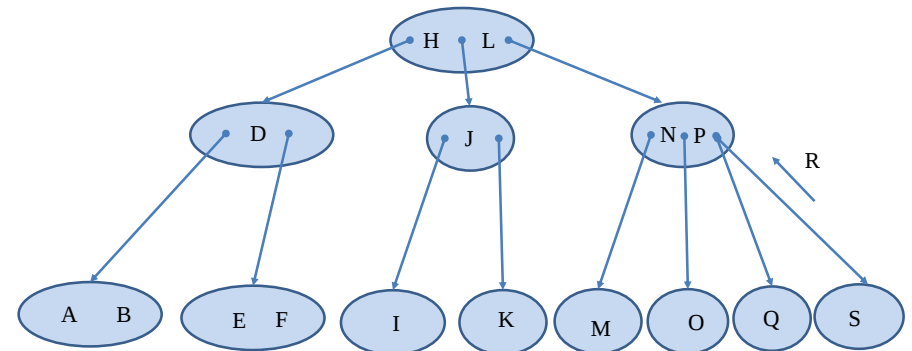


S overflows
this node

S overflows this node

39

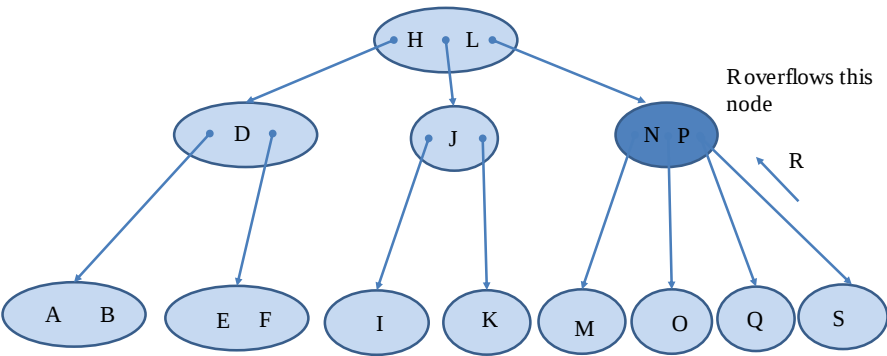
Example: insert S



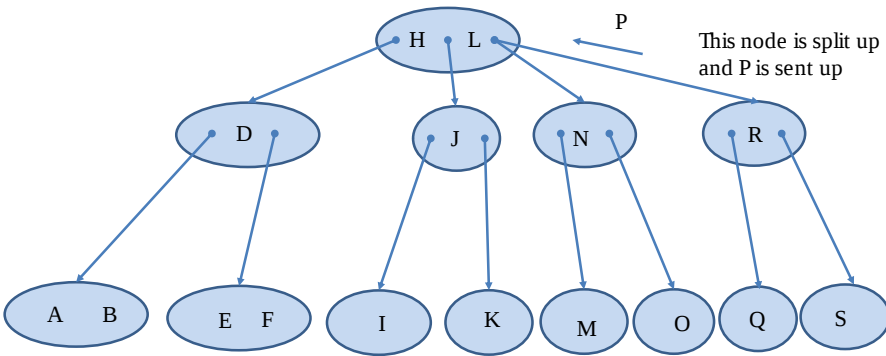
This node is split
and R is sent up

40

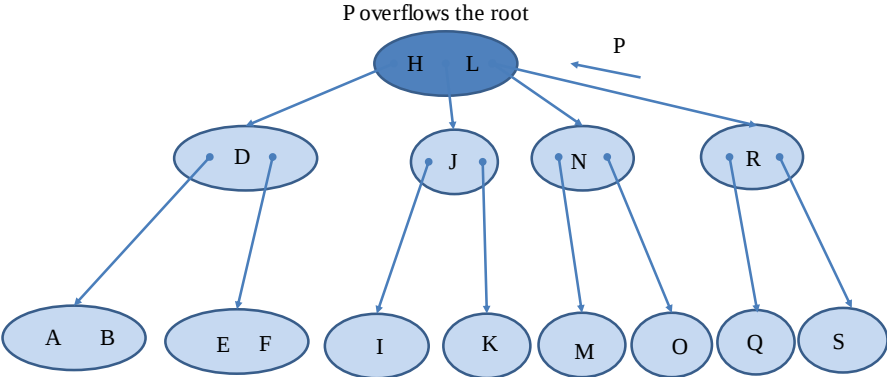
Example: insert S



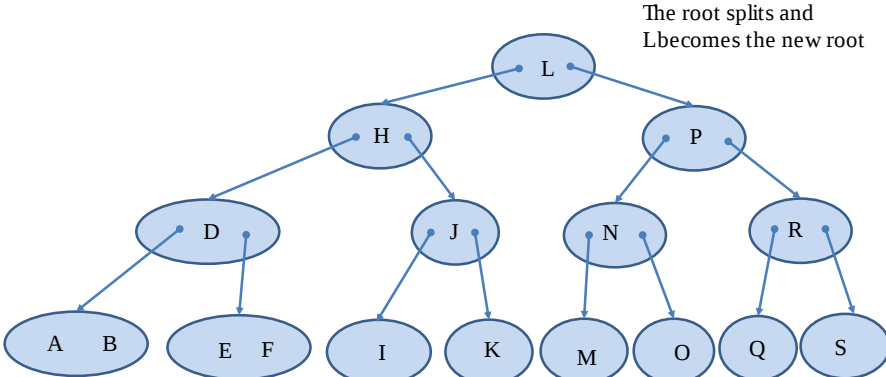
Example: insert S



Example: insert S



Example: result



Complexity of insertion

- We traverse the tree
 - From the root to a leaf when searching
 - From the leaf back to the root while splitting
- Each split takes constant time
 - We do at most $h + 1$ of them
- So in total $O(h) = O(\log n)$ steps
 - Recall, the tree is balanced

45

2-4 trees

- A **2-4 tree** (or 2-3-4 tree) is a 4-way search tree with 2 extra properties
- **Size property**
 - Each node contains between **1 and 3 values**
 - **Internal** nodes with n values have exactly $n + 1$ **children**
- **Depth property**
 - All **leaves** have the **same depth** (lie on the same level)
- Such trees are **balanced**
 - $h = O(\log n)$
 - Proof: exercise

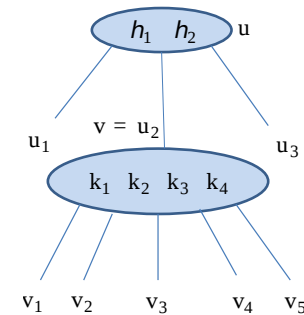
46

Insertion in 2-4 trees

- Same as for 2-3-trees
 - Search for the value
 - Insert at a leaf
- In case of an overflow (5-node)
 - Split it into a 3-node and a 2-node
 - Move the separator value k_3 to the parent

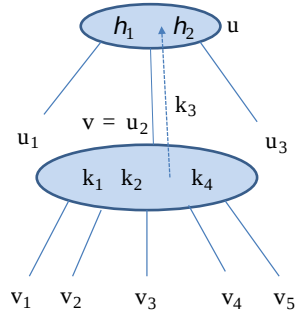
47

Overflow at a 5-node



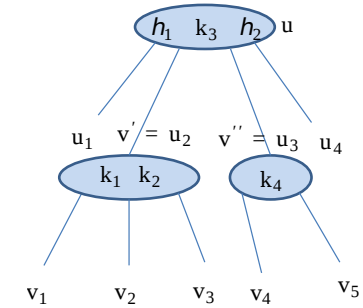
48

The separating value is sent to the parent node



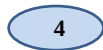
49

Node replaced with a 3-node and a 2-node



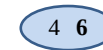
50

Example: insert 4



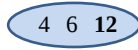
51

Example: insert 6



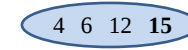
52

Example: insert 12



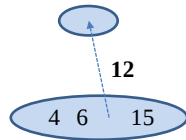
53

Example: insert 15 - overflow



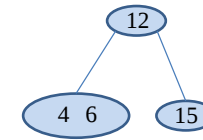
54

Creation of new root node



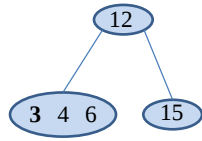
55

Split



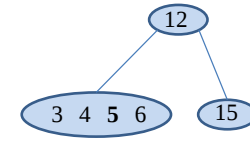
56

Example: insert 3



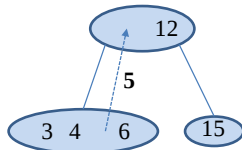
57

Example: insert 5 - overflow



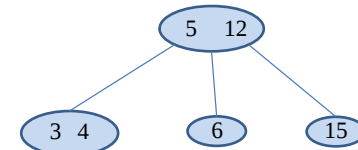
58

5 is sent to the parent node



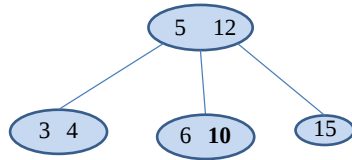
59

Split



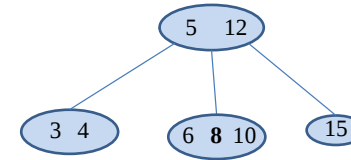
60

Example: insert 10



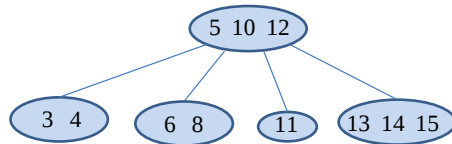
61

Example: insert 8



62

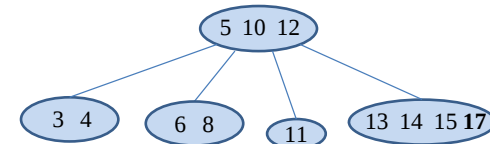
Example



Inserted 11, 13 and 14.

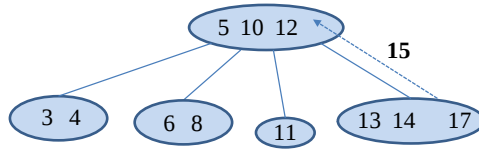
63

Example: insert 17 - overflow



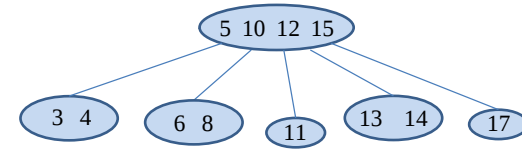
64

Split and send 15 to the parent node



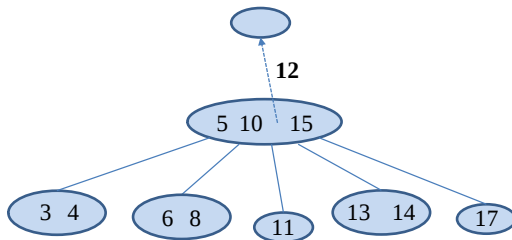
65

The root overflows



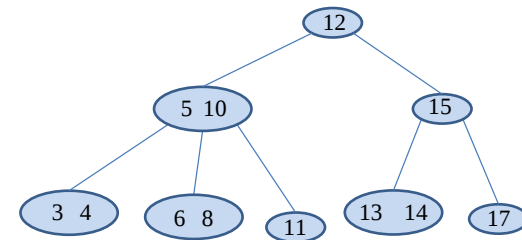
66

Creation of new root



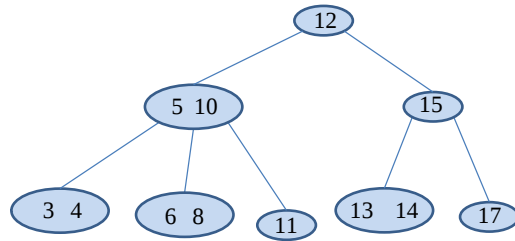
67

Split



68

Final tree



69

Complexity

- Same as for 2-3-trees
 - At most h splits
 - Each split is constant time
- $O(\log n)$
 - Because the tree is balanced

70

Removal in 2-4 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
 - Similar to the BST case of nodes with two children
- To remove a value from a **leaf**
 - We simply remove it
 - But it might violate the **size** property (**underflow**)

71

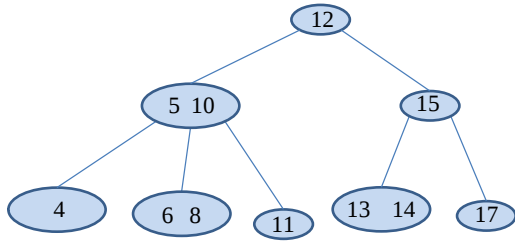
Fixing underflows

Two strategies for fixing an underflow at ν

- Is there an **immediate sibling** w with a “spare” value? (2 or 3 values)
- 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

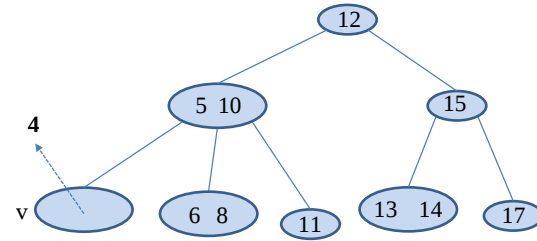
72

Initial tree



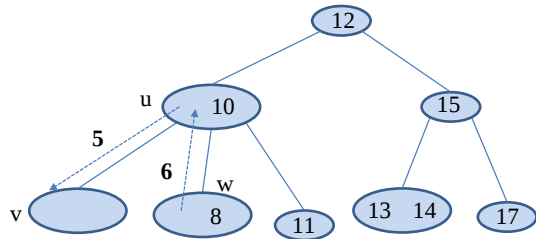
73

Remove 4



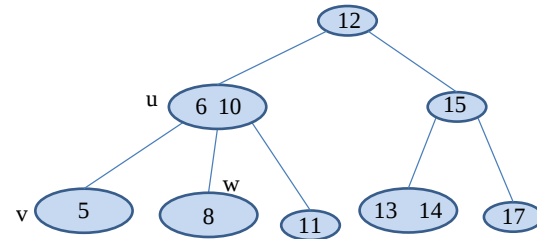
74

Transfer



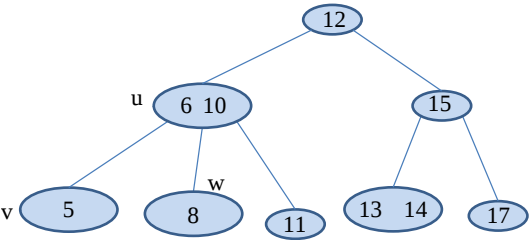
75

After the transfer

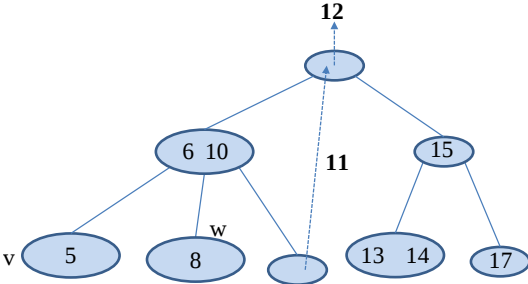


76

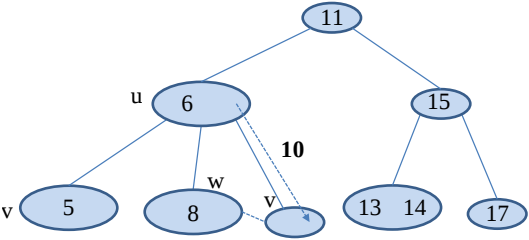
Remove 12



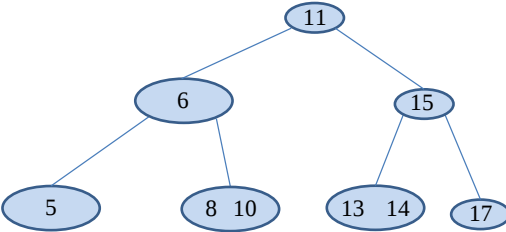
Remove 12



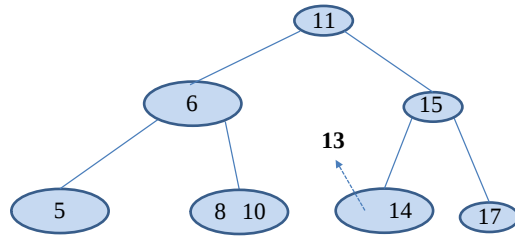
Fusion of and



After the fusion

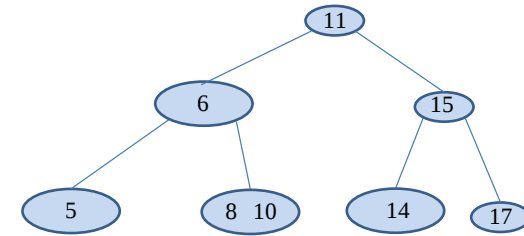


Remove 13



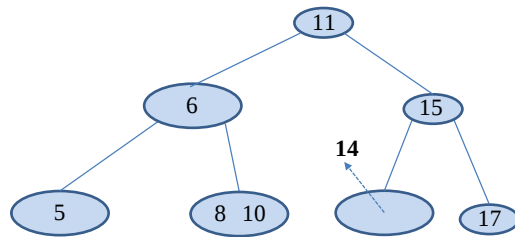
81

After the removal of 13



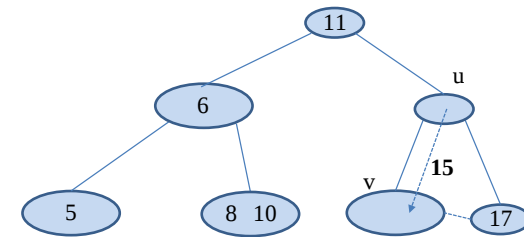
82

Remove 14 - underflow



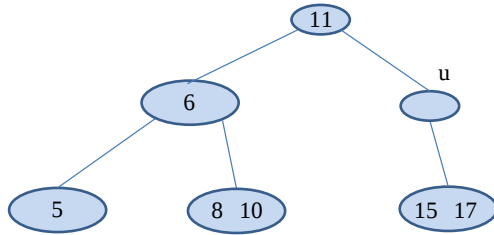
83

Fusion



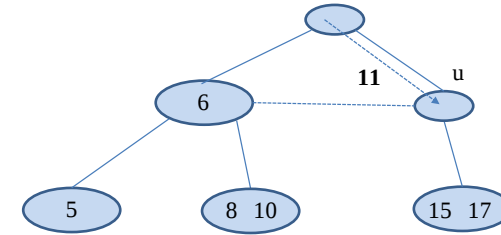
84

Underflow at



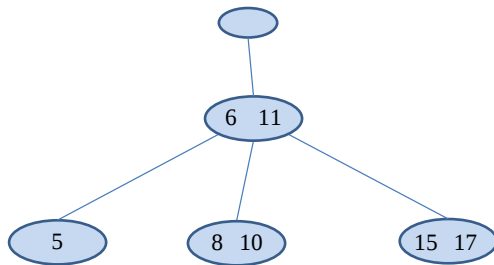
85

Fusion



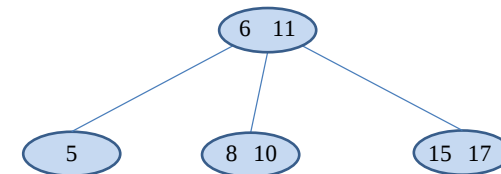
86

Remove the root



87

Final tree



88

Readings

- T. A. Standish. *Data Structures, Algorithms and Software Principles in C*. Section 9.9
- M. T. Goodrich, R. Tamassia and D. Mount. *Data Structures and Algorithms in C++*. Section 10.4
- R. Sedgewick. *Αλγόριθμοι σε C*. 3η Αμερικανική Έκδοση. Εκδόσεις Κλειδάριθμος. Section 13.3