15CSE201 : Data Structures and Algorithms

Lecture 13 :Multiway Search Trees

Ritwik M

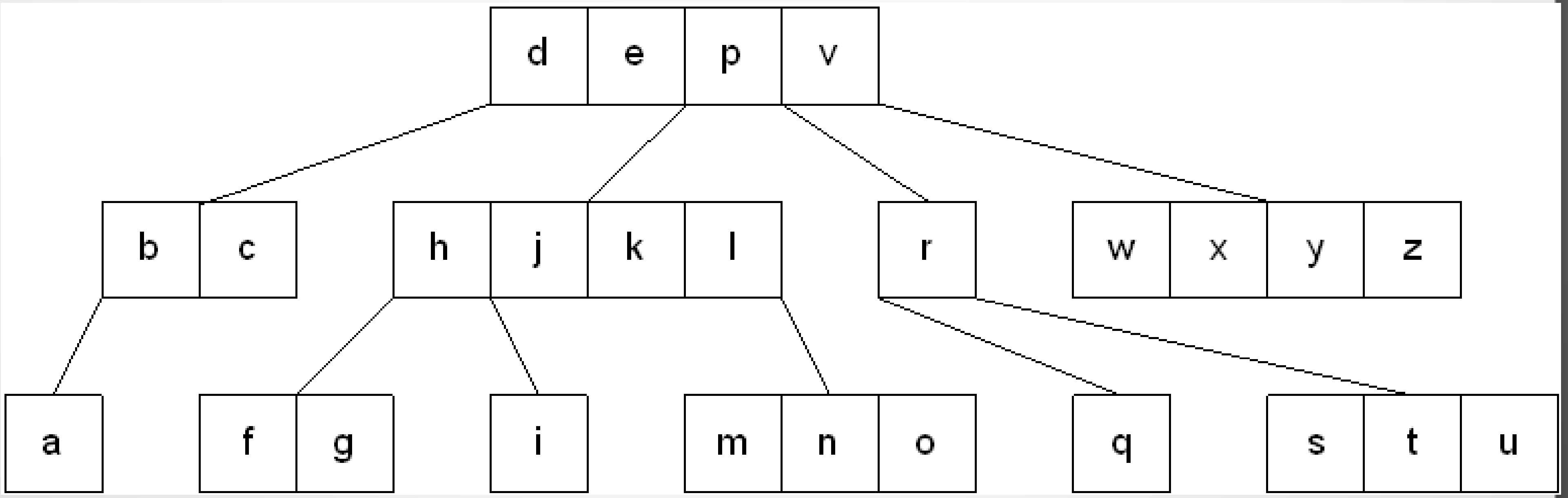
Based on the reference materials by Prof. Goodrich and Dr. Vidhya Balasubramanian

Contents

- Recap
- The importance of being balanced
- AVL trees
  - Definition and balance
  - Rotations
  - Insertion

Multiway Search Trees

- Trees whose internal nodes have two or more children
- Eg. Eg: 2,4 tree, red-black tree, B-Tree

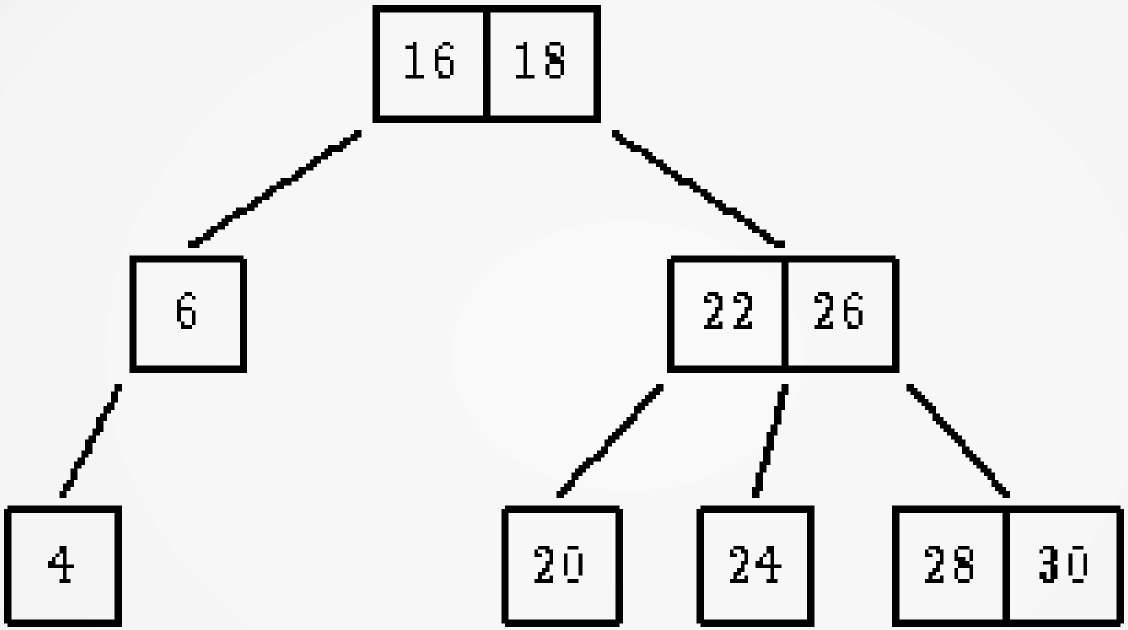


(source: http://faculty.cs.niu.edu/~freedman/340/340notes/gifImages/340multi1.gif)

Understanding M-Way Search

- An m-way search tree is a tree in which
  a. The nodes hold between 1 to m-1 distinct keys
  b. The keys in each are sorted
  c. A node with k values has k+1 subtrees, where the subtrees may be empty.
  d. The i'th subtree of a node $[v_1, \ldots, v_k]$, $0 \leq i \leq k$, may hold only values v in the range $v_i \leq v \leq v_{i+1}$ (v_0 is assumed to equal $-\infty$ and v_{k+1} is assumed to equal *infinity*)

Example

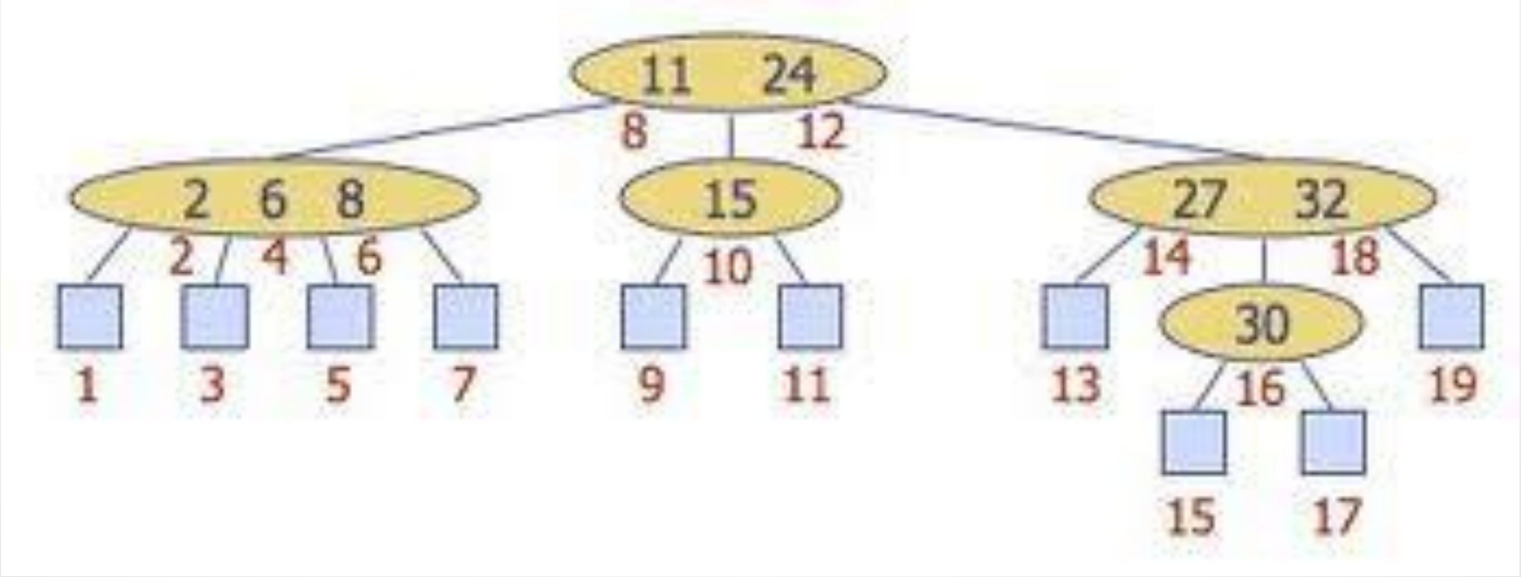


Another manner of understanding m-way search trees

- A multi-way search tree is an ordered tree such that:
  - Each internal node has at least two children and stores d-1 key-element items (k_i, o_i), where d is the number of children
- For a node with children v_1 v_2 ... v_d storing keys k_1 k_2 ... k_{d-1}
  - keys in the subtree of v_1 are less than k_1
  - keys in the subtree of v_i are between k_{i-1} and k_i
  - keys in the subtree of v_d are greater than k_{d-1}

Multi-Way Inorder Traversal

- Visit item (k_i, o_i) of node v between the recursive traversals of the subtrees of v rooted at children v_i and v_{i+1}
- An inorder traversal of a multi-way search tree visits the keys in increasing order

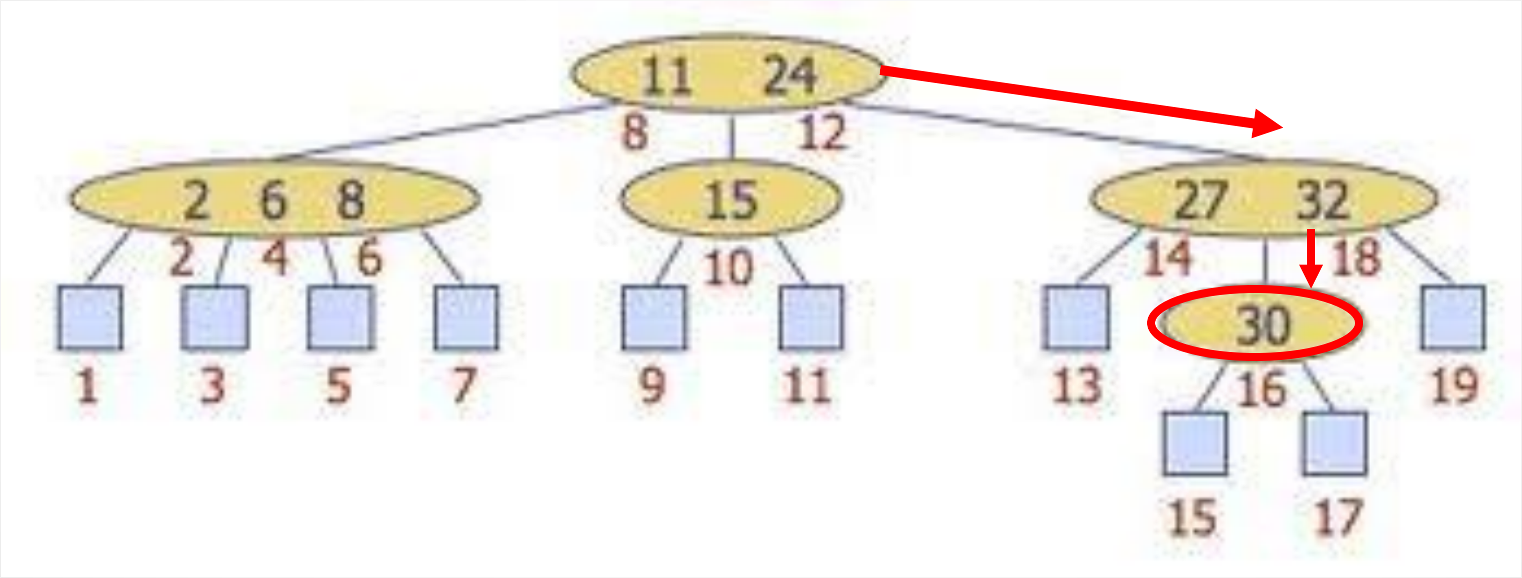


Multi-way tree: Searching

- Similar to search in binary tree
- At each internal node with children v_1 v_2 ... v_d storing keys k_1 k_2 ... k_{d-1}
  - If $k = k_i$ (i = 1, ..., $d - 1$): search terminates successfully
  - If $k = k_1$: continue search in child v_1
  - If $k_{i-1} < k$ (i = 2, ..., $d - 1$): continue search in child v
  - If $k > k$d : continue search in child v_d
- Terminating at an external node terminates the search

Multi-way Searching: Example

Search for 30

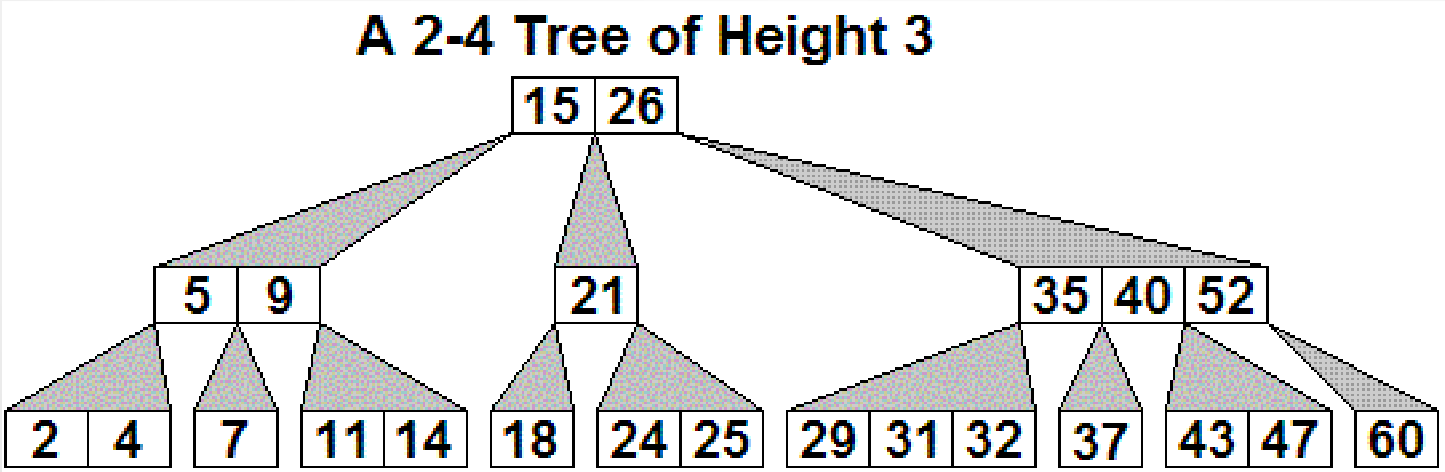


(2,4) Trees

- 2-4 or 2-3-4 trees
  - Keeps primary tree balanced
  - Secondary data structures stored at each node is small
- Properties
  - Node-Size Property
    - every internal node has at most four children
  - Depth Property
    - all the external nodes have the same depth
- Depending on the number of children, an internal node of a (2,4) tree is called a 2-node, 3-node or 4-node
- Note: - Lack of single children

(2,4) trees

- Each internal node has either
  - two children (2-node) and one data element
  - three children (3-node) and two data elements or
  - four children (4-node) and three data elements



Src:anh.cs.luc.edu

(2,4) trees

- A node can contain 1 / 2 / 3 entries
- Example:

1	12	18

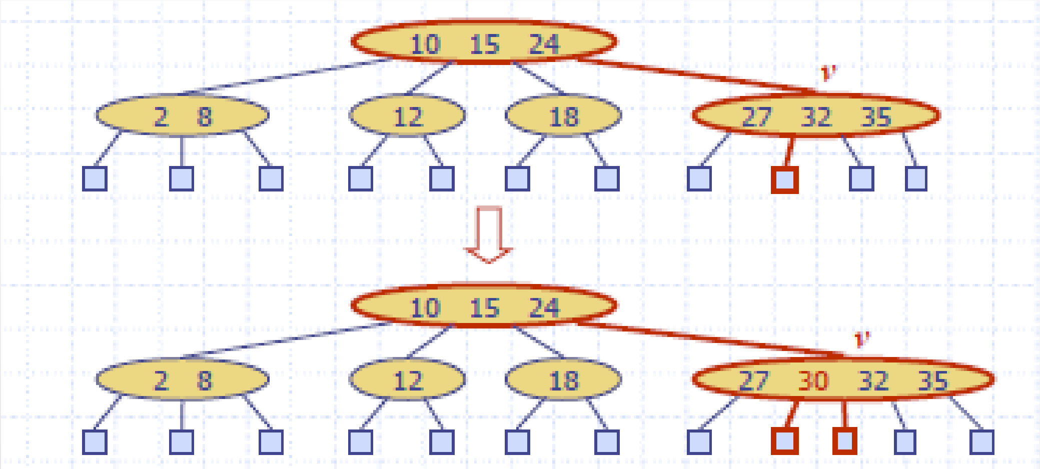
- How many children are allowed for the above node?

Height of a (2,4) tree

- Theorem: Height of a (2,4) tree storing n items is Q(log*n*)
- Proof
  - Let *h* be the height of a (2,4) tree with *n* items
  - Since there are at least 2i items at depth i = 0, … , h − 1 and no items at depth h, we have
    - *n* ≥ 1 + 2 + 4 + … + 2h−1 = 2h − 1
  - Thus, *h* ≤ log (*n* + 1)
- Searching an item in a (2,4) tree takes O(log*n*) time

Insertion

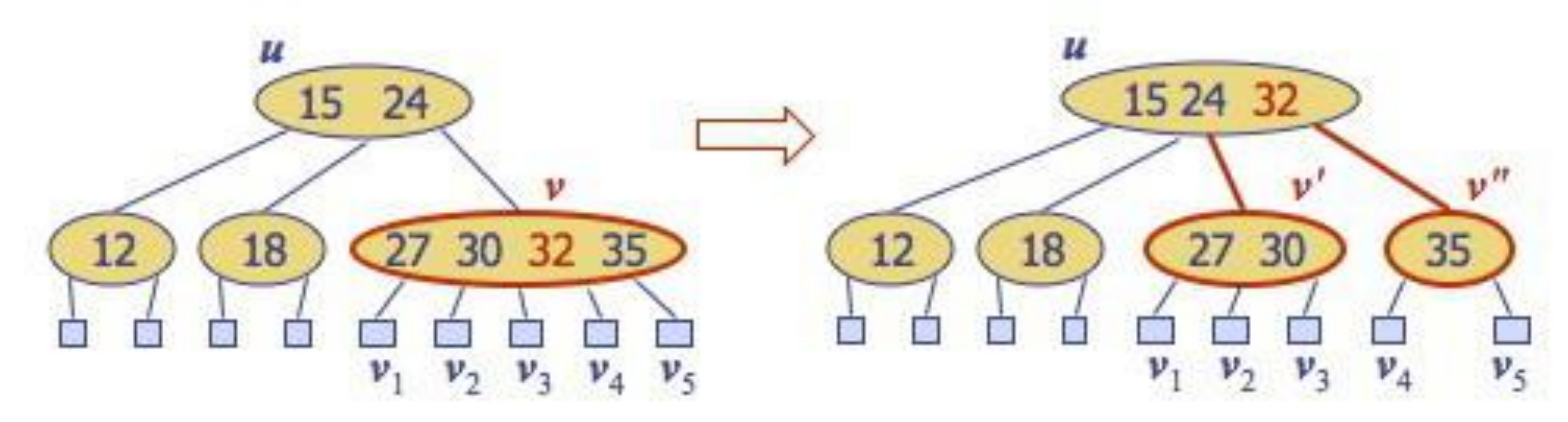
- Insert a new item (k, o)
- Let *v* be node reached when searching for *k*
- Insert node at v
  - Will preserve depth property
  - May cause overflow
    - Node is 5-node (has 4 keys)



Note: if insert() encounters a 3 key node, the middle key is placed in parent node

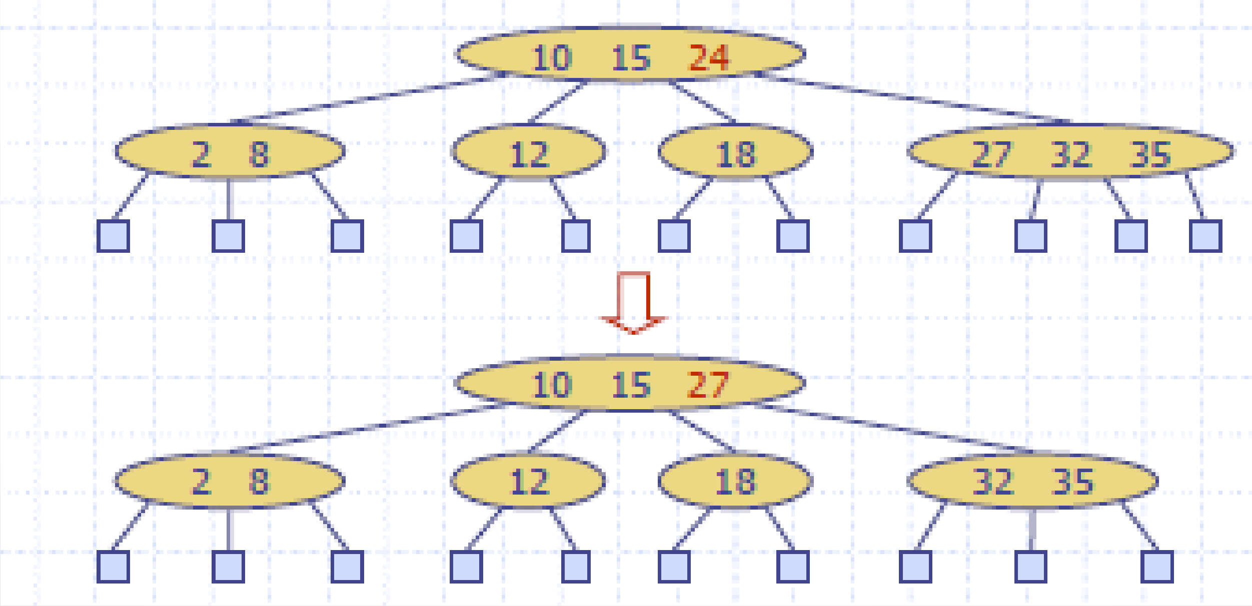
Insertion : Example

- Cost of insertion depends on number of splits
  - Cost of each split (constant)
  - Atmost log*n* splits required



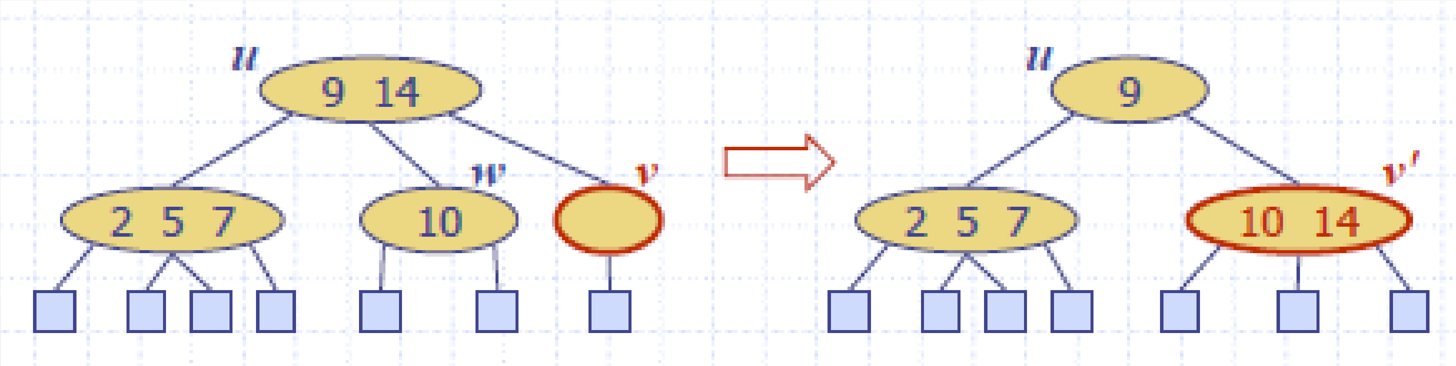
Deletion

- Reduce the deletion to the case where item is at leaf
  - If item is not at a leaf, replace it with it its inorder successor (or, equivalently, with its inorder predecessor)
  - e.g to delete 24, replace with 27



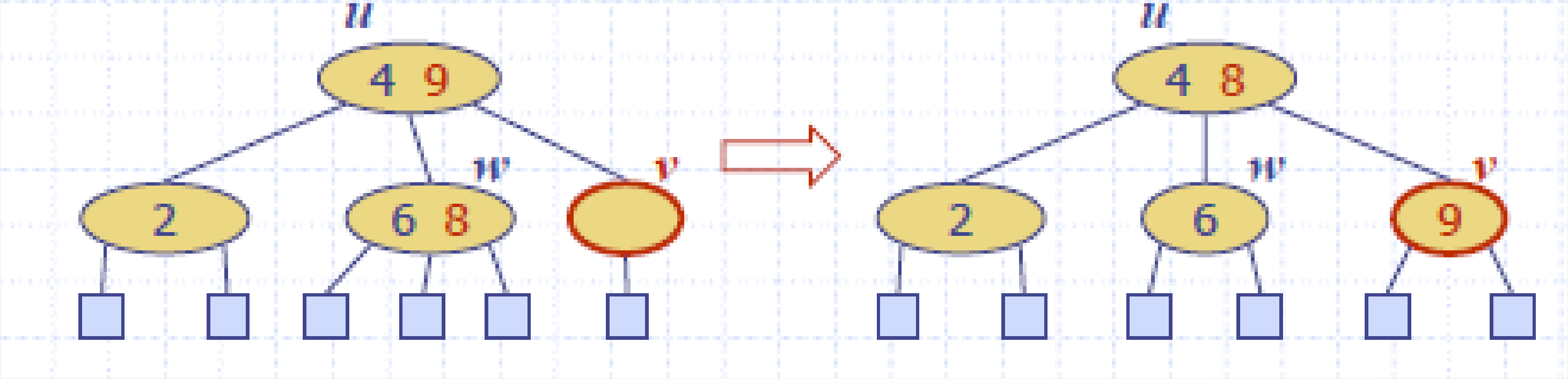
Underflow and Fusion

- Deleting an item from a node v may cause an underflow, where node v becomes a 1-node with one child and no keys
- Handling an underflow at node *v* with parent *u*
  - Case 1: the adjacent siblings of *v* are 2-nodes
  - Fusion operation: merge v with an adjacent sibling w and move an item from u to the merged node v‘
  - The underflow may propagate to the parent u

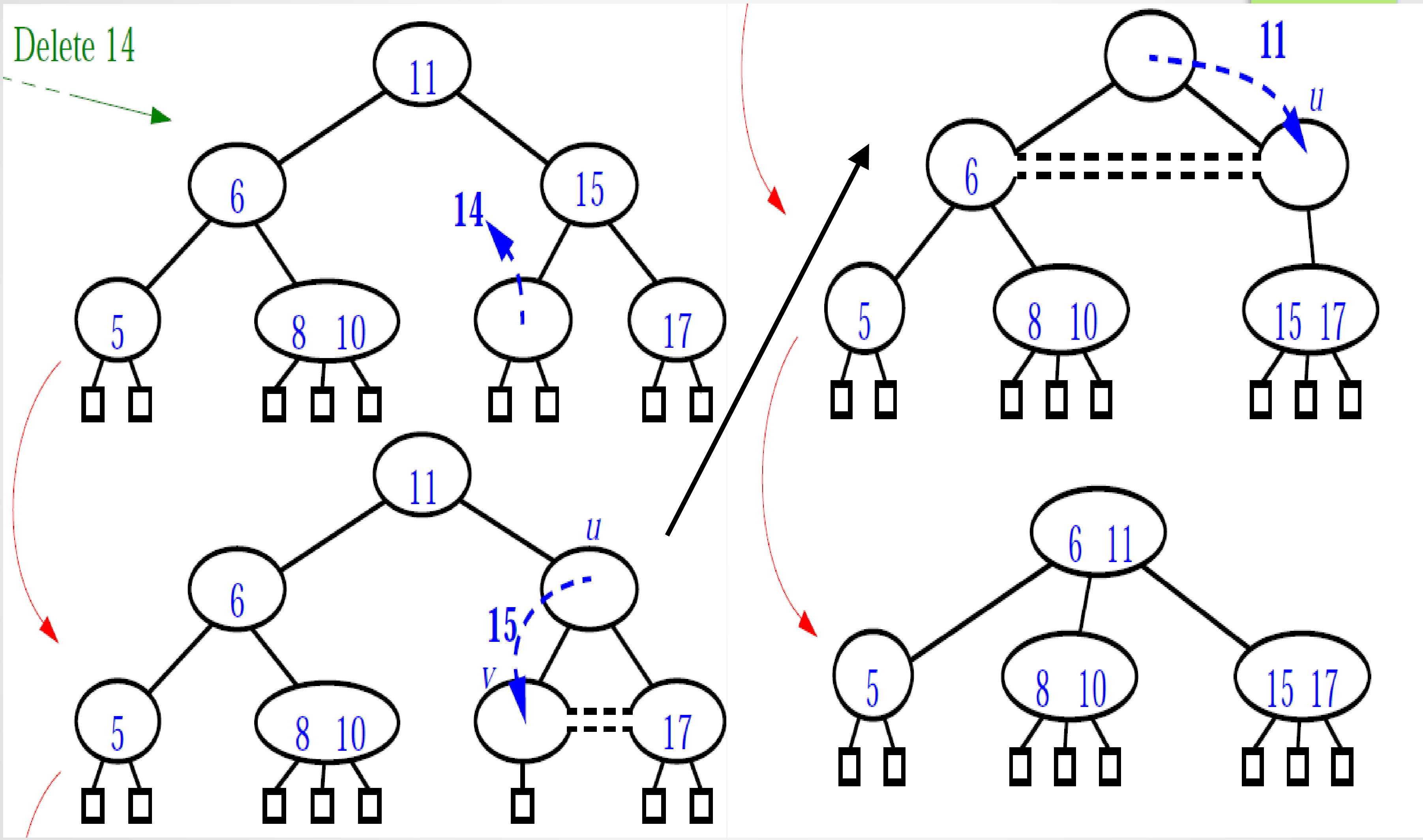


Underflow and Transfer

- Case 2: an adjacent sibling *w* of *v* is a 3-node or a 4-node
  - Transfer operation:
    - Move a child of w to v
    - Move an item from u to v.
    - Move an item from w to u
  - After a transfer, no underflow occurs



Example of Underflow Cascade



Exercise

- Consider the following sequence of keys. Insert them into an initially empty (2,4) tree in order.
  - 5,16,22,45,2,10,18,30,50,12,1
- Show the effect of insertion of the following values in the above tree
  - 75, 9, 60, 56
- Delete the following nodes
  - 16, 30, 56

(a,b) Trees

- Generalization of 2-4 trees
- Size of nodes and running time of operations depend on a and b
  - Each node has between *a* and *b* children
  - Stores between *a*-1 and *b*-1 keys
  - 2 ≤ a ≤ (b1)/2
- All external nodes have same depth.
- Height of an (a,b) tree storing n items is O(log*n*/log*a*)

B-Trees

- Version of (a,b) tree which is best method for storing indexes in external memory
  - B-trees keep related records (that is, records with similar key values) on the same disk block, which helps to minimize disk I/O on searches due to locality of reference.
- A B-Tree of order d is an (a,b) tree with a = [d/2], and b = d
  - Each internal node, except for the root, has between [d/2] and d children
  - Underflow is when the number of children in a node is < d/2
  - Overflow occurs when the number of children in a node > d
  - Height-balanced and all children at same level
  - Root has atleast 2 children or is a leaf
  - 2-4 tree is B-Tree of order 4

B+Trees

- Stores values only at the leaf nodes
- Internal nodes store key values, but these are used solely as placeholders to guide the search
  - store keys to guide the search, associating each key with a pointer to a child B+-tree node