CSE 201: Data Structures

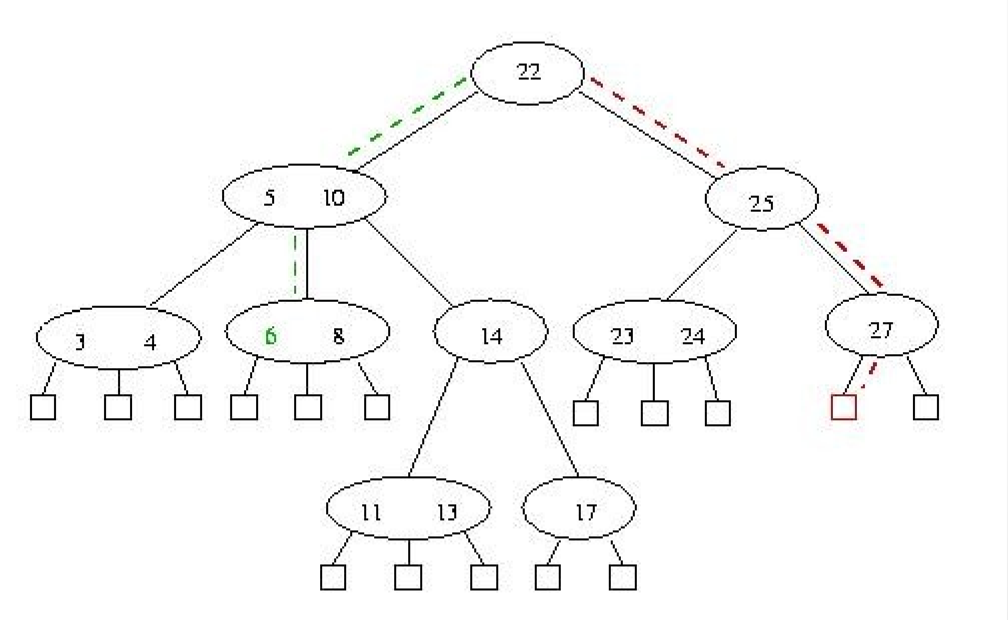
Lecture 8.3: Multi-way Search Trees

Dr. Vidhya Balasubramanian

Amrita School of Engineering
Amrita Vishwa Vidyapeetham

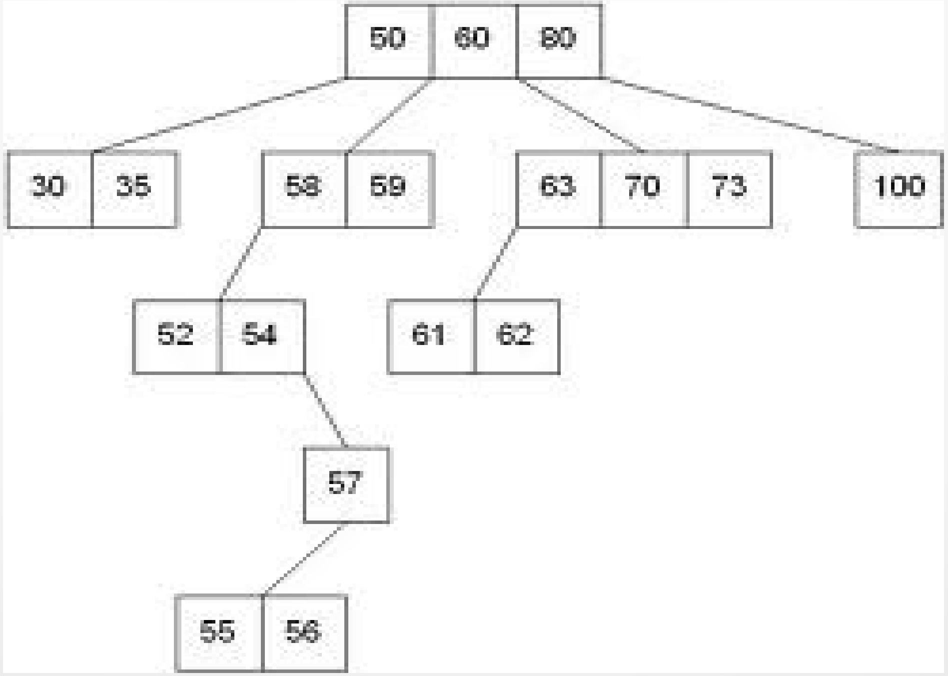
Multi-Way Search Trees

- Multi-Way search trees
  - Trees whose internal nodes have two or more children
  - e.g
    - 2,4 tree, red-black tree, B-Tree



Multi-Way Search Trees: Philosophy

- A key stored in the subtree of T rooted at a childnode v_i must be “in between” two keys stored at *v*
  - A *d*-node stored *d*-1 regular keys
  - Forms the basic of algorithm for searching in a multiway search tree



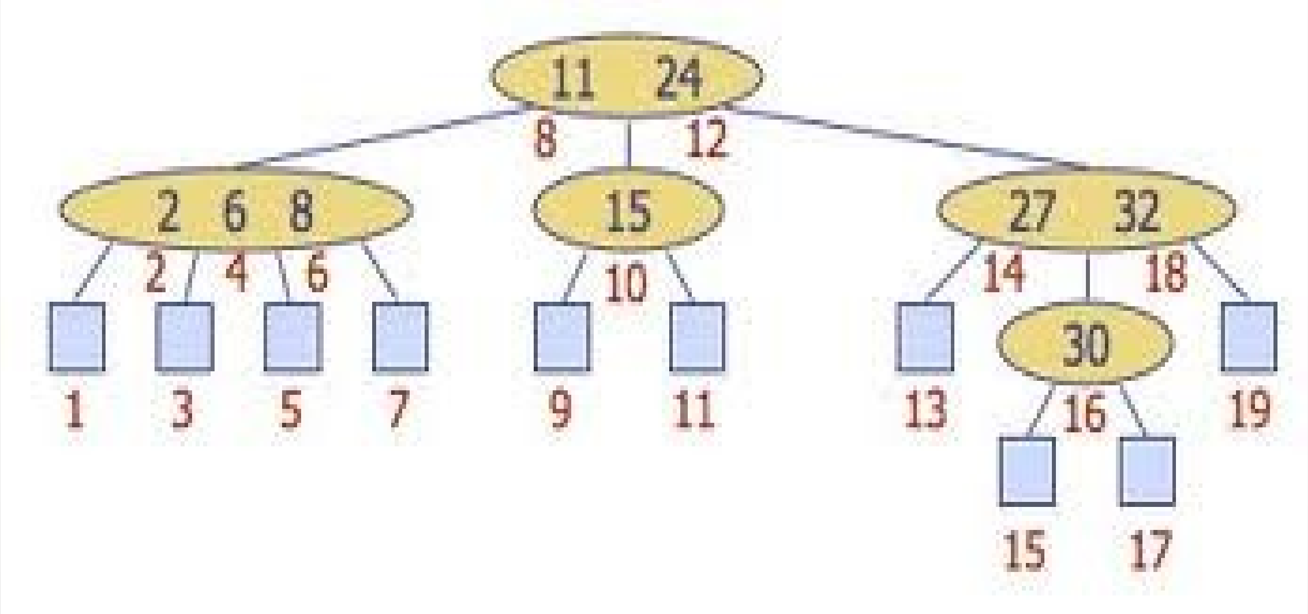
faculty.cs.niu.edu

Multi-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 $v_1\ v_2\ \ldots\ v_d$ storing keys $k_1\ k_2\ \ldots\ k_{d-1}$
  - 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
    - $i = 2, \ldots, d - 1$
  - 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

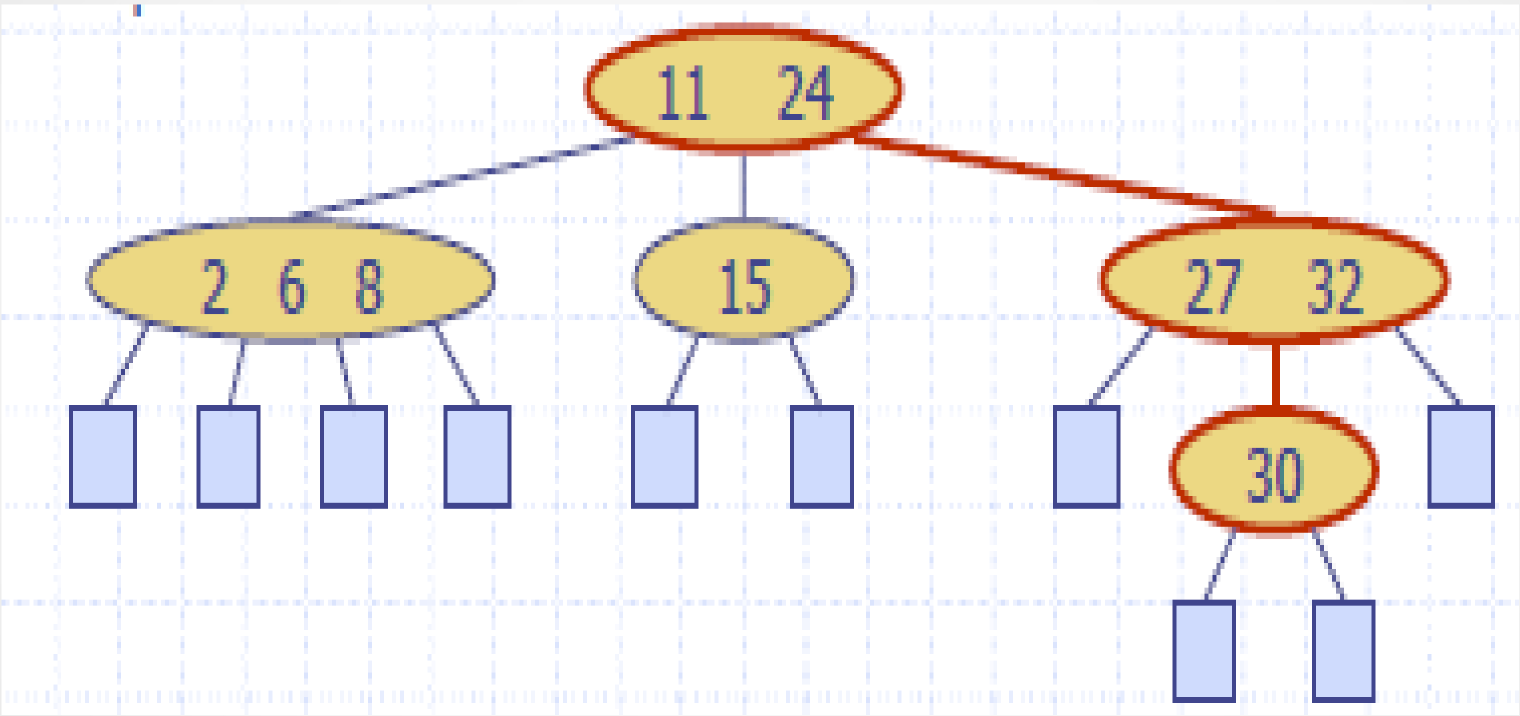


Multi-way tree: Searching

- 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_i
  - 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

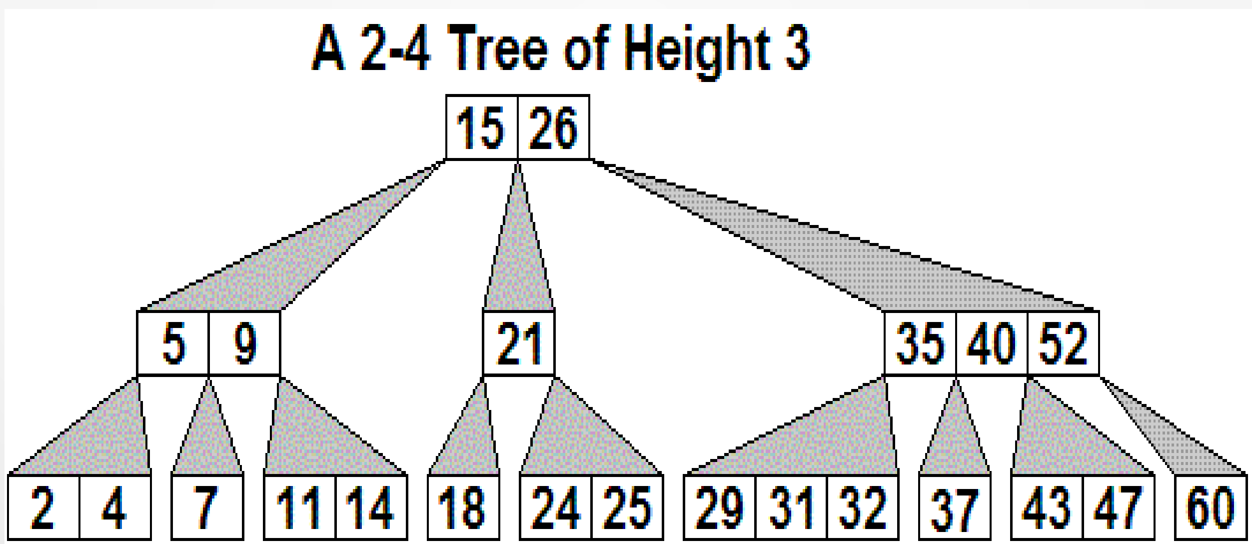


(2,4) Trees

- Multi-way search trees also called 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

(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.



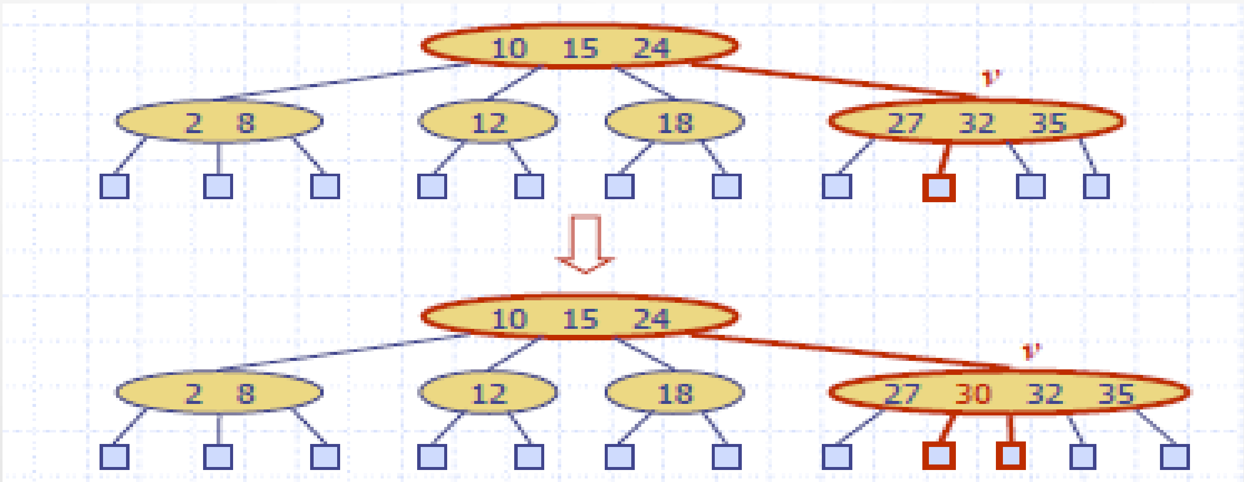
Src:anh.cs.luc.edu

Height of a (2,4) tree

- Theorem: Height of a (2,4) tree storing n items is $\Theta(\log n)$
- Proof
  - Let h be the height of a (2,4) tree with n items
  - Since there are at least 2^i items at depth $i = 0, \ldots, h - 1$ and no items at depth h, we have
    - $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} = 2^h - 1$
  - Thus, $h \leq \log (n + 1)$
- 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)

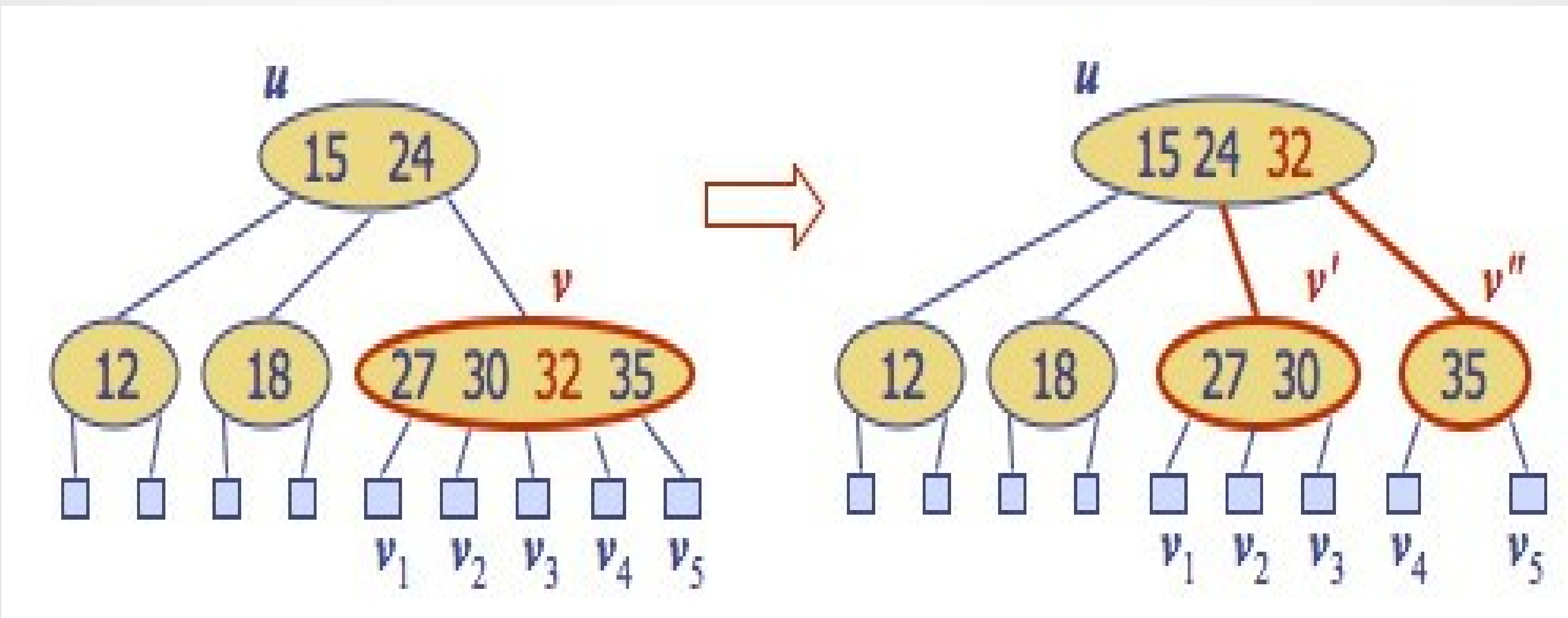


Overflow and Split

- We handle an overflow at a 5-node v with a split operation:
  - let $v_1 \ldots v_5$ be the children of v and $k_1 \ldots k_4$ be the keys of v
  - node v is replaced with nodes v' and v"
    - v' is a 3-node with keys k_1, k_2 and children v_1, v_2, v_3
    - v'' is a 2-node with key k_4 and children v_4, v_5
  - key k_3 is inserted into the parent u of v (a new root may be created)
- The overflow may propagate to the root

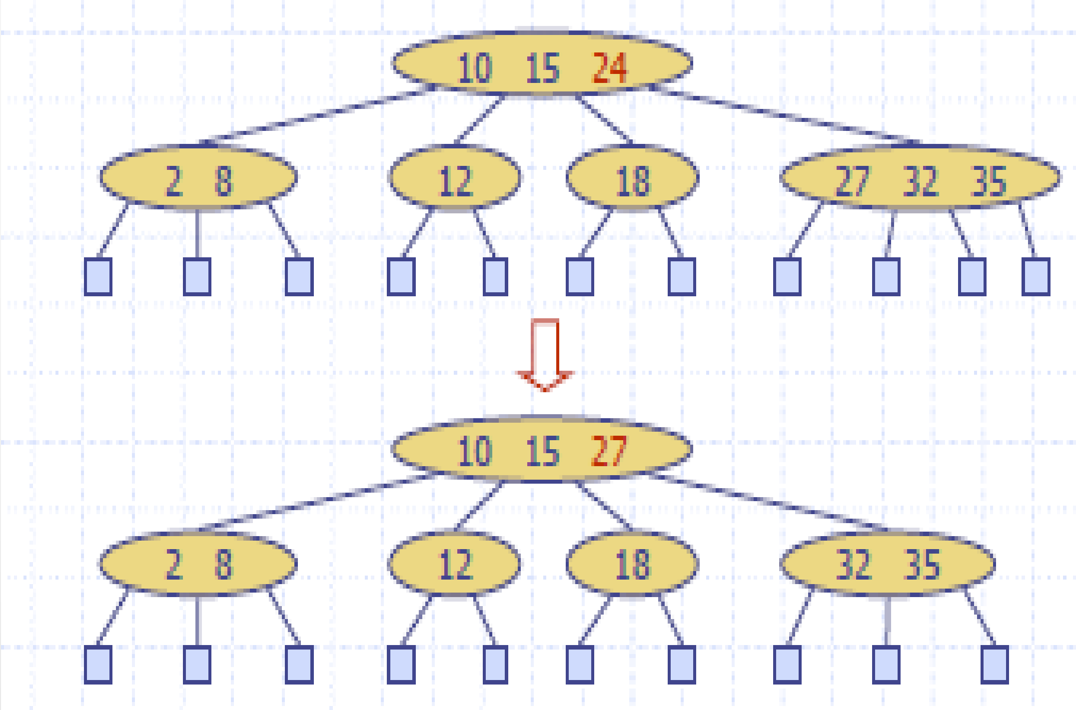
Insertion : Example

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



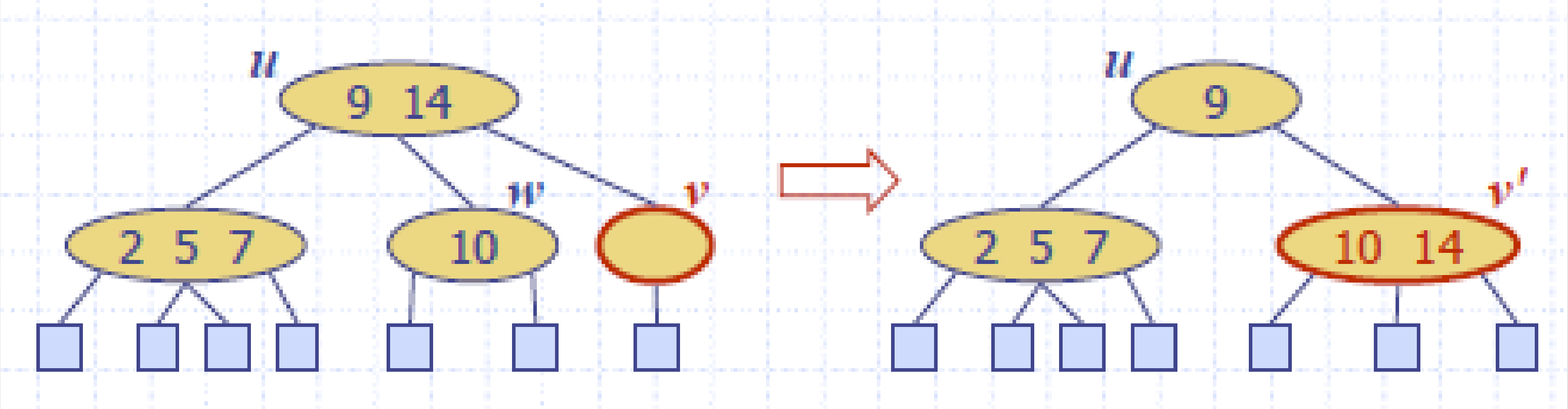
Deletion

- 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



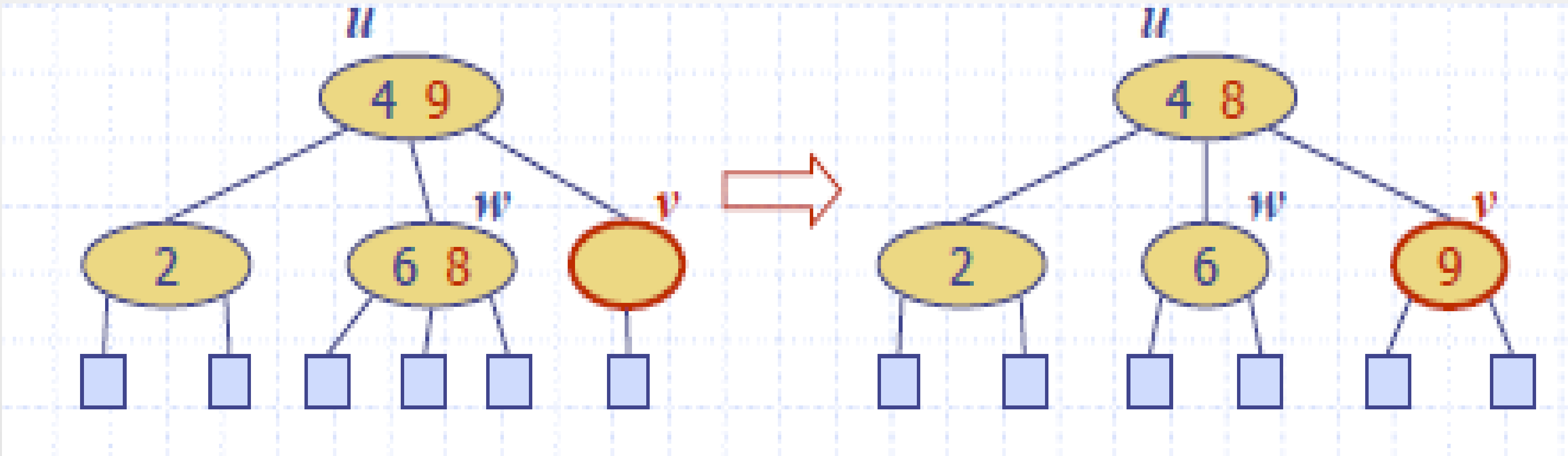
Underflow and Fusion

- 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

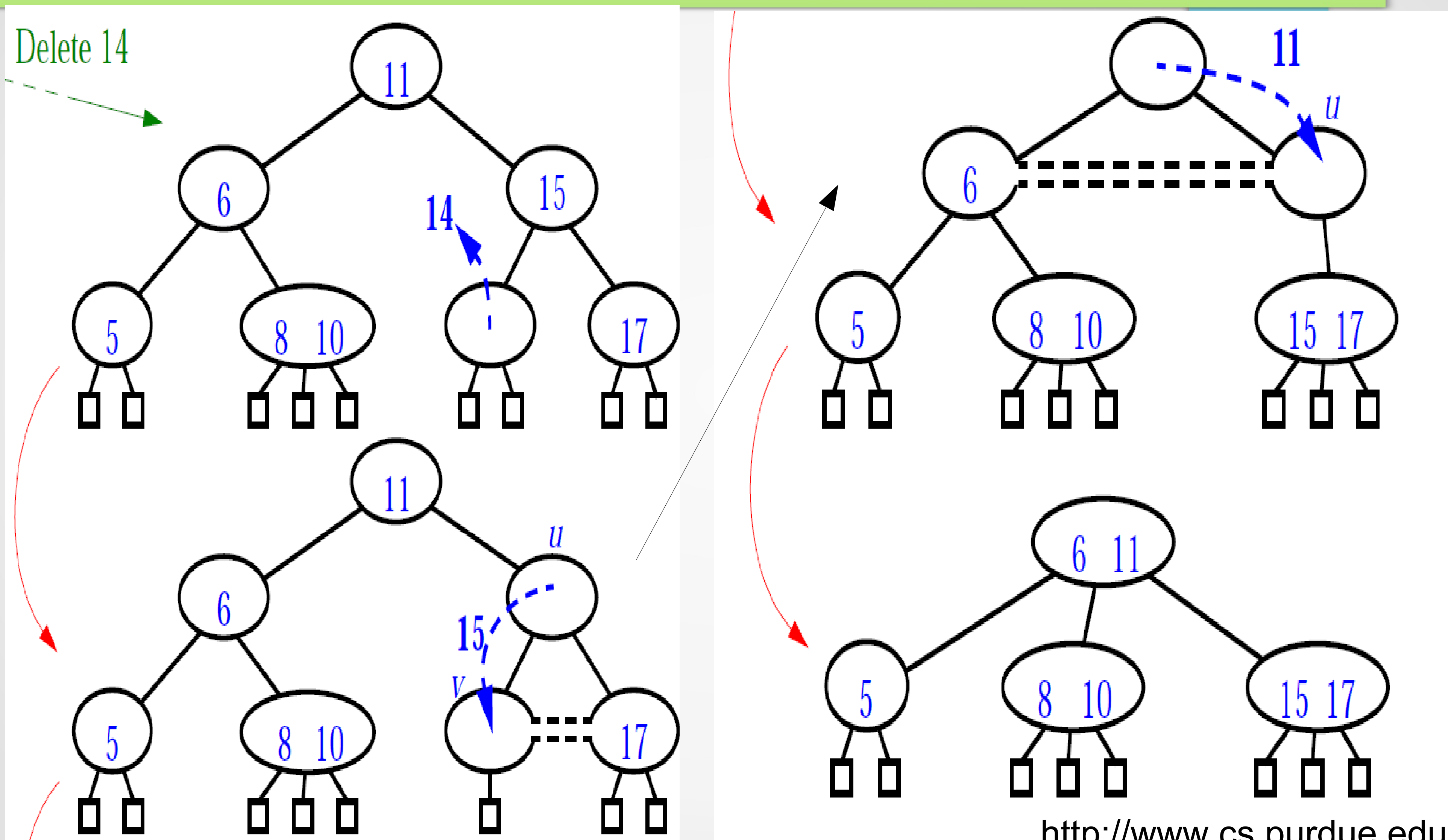


Underflow and Transfer

- 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



http://www.cs.purdue.edu/homes/ayg/CS251/slides/chap13a.pdf

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 values
  - 16, 30, 56

(a,b) Trees

- Generalization of 2-4 trees
  - Each node has between *a* and *b* children
  - Stores between *a*-1 and *b*-1 keys
  - $2 \leq a \leq (b1)/2$
- Size of nodes and running time of operations depend on *a* and *b*
- 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

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