

Advanced Data Structure and Algorithm

2-4 (2-3-4) Trees

(2,4) Trees

- What are they?
 - They are search Trees (but not binary search trees)
 - They are also known as 2-4, 2-3-4 trees

That's a very nice hat.

That's not a hat!
That's my head!
I'm *Tree* Head!

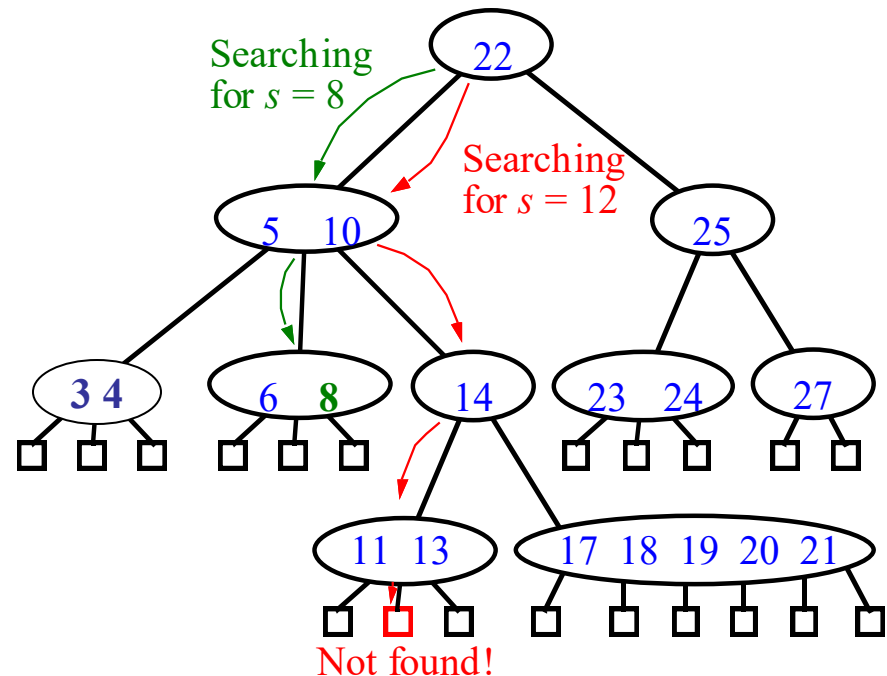


Multi-way Search Trees

- Each internal node of a multi-way search tree T :
 - has at least two children
 - stores a collection of items of the form (k, x) , where k is a key and x is an element
 - contains $d - 1$ items, where d is the number of children
 - “contains” 2 pseudo-items: $k_0 = -\infty$, $k_d = \infty$
- Children of each internal node are “between” items
- all keys in the subtree rooted at the child fall between keys of those items
- External nodes are just placeholders

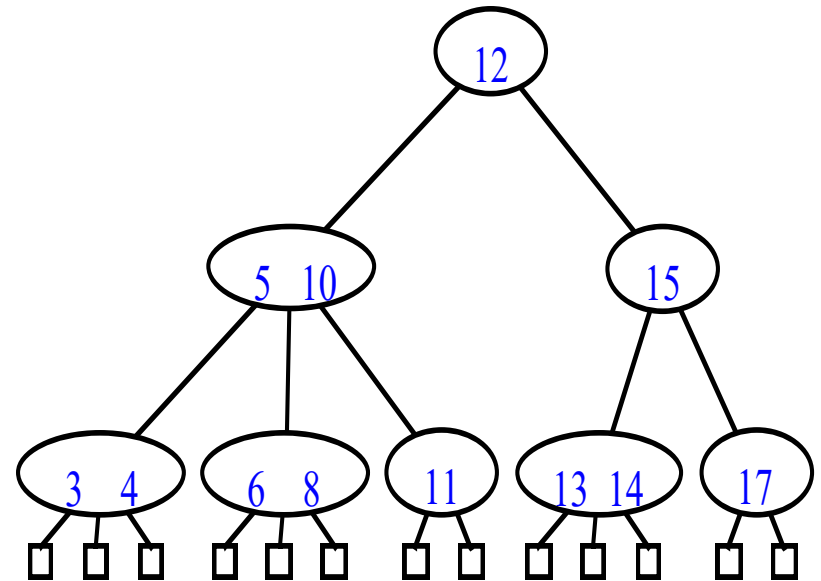
Multi-way Searching

- Similar to binary searching
 - If search key $s < k_1$ search the leftmost child
 - If $s > k_{d-1}$, search the rightmost child
- That's it in a binary tree; what about if $d > 2$?
 - Find two keys k_{i-1} and k_i between which s falls, and search the child v_i .
- What would an in-order traversal look like?



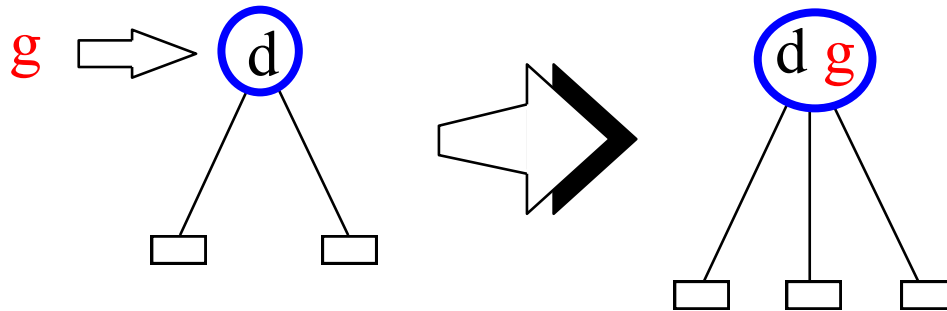
(2,4) Trees

- Properties:
 - At most 4 children
 - All external nodes have same depth
 - Height h of (2,4) tree is $O(\log n)$.
- How is the last fact useful in searching?

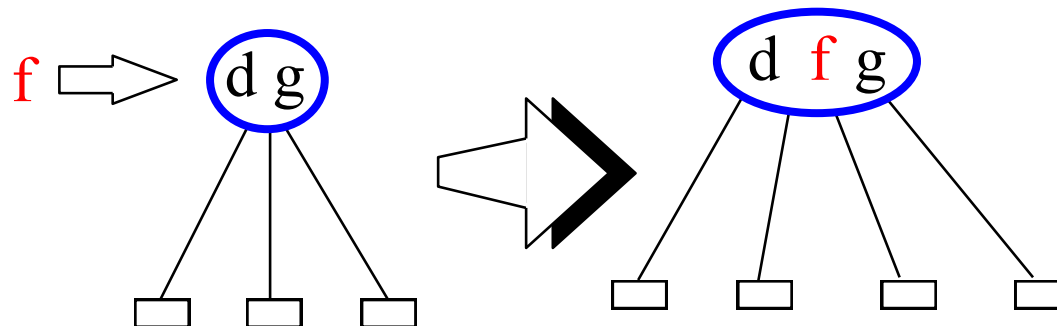


Insertion into (2,4) Trees

- Insert the **new key** at the **lowest internal node reached** in the search
- **2-node** becomes **3-node**



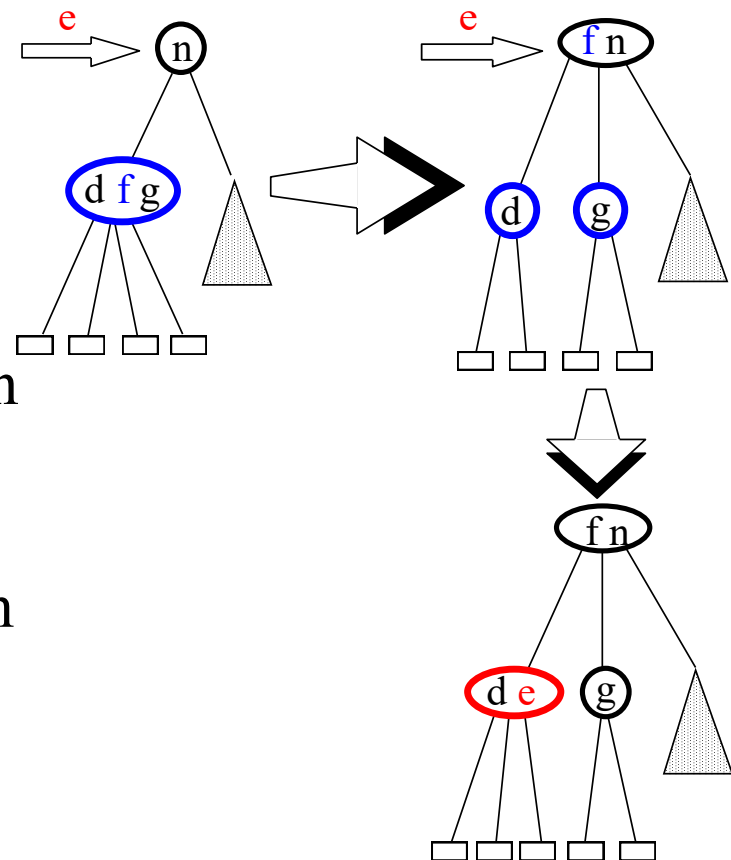
- **3-node** becomes **4-node**



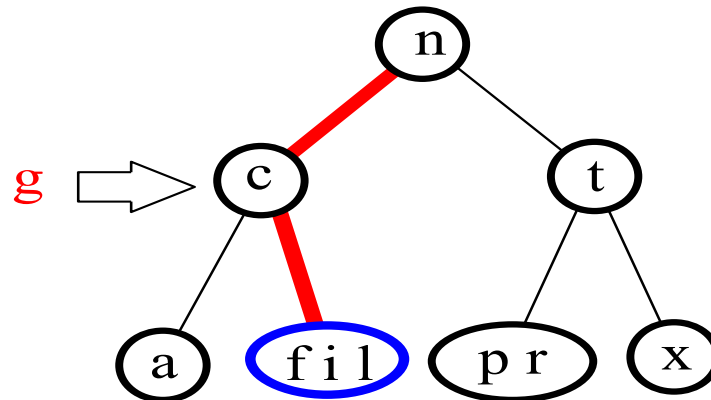
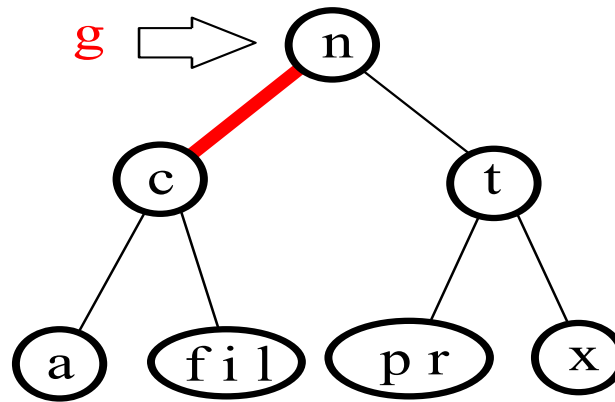
- What about a **4-node**?
- **We can't insert another key!**

Top Down Insertion

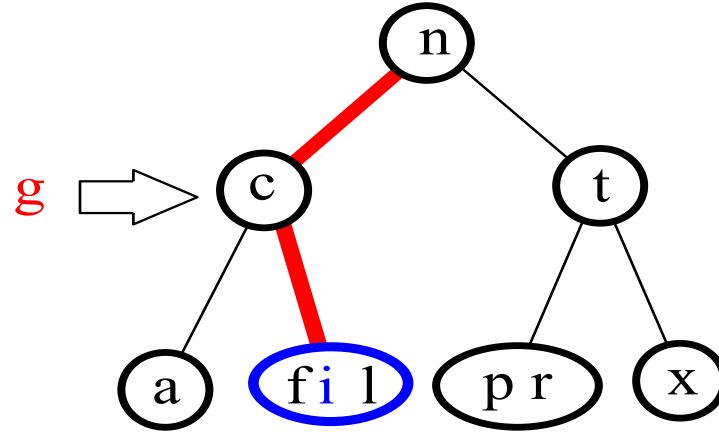
- In our way down the tree, whenever we reach a 4-node, we break it up into two 2-nodes, and move the middle element up into the parent node
- Now we can perform the insertion using one of the previous two cases
- Since we follow this method from the root down to the leaf, it is called *top down insertion*



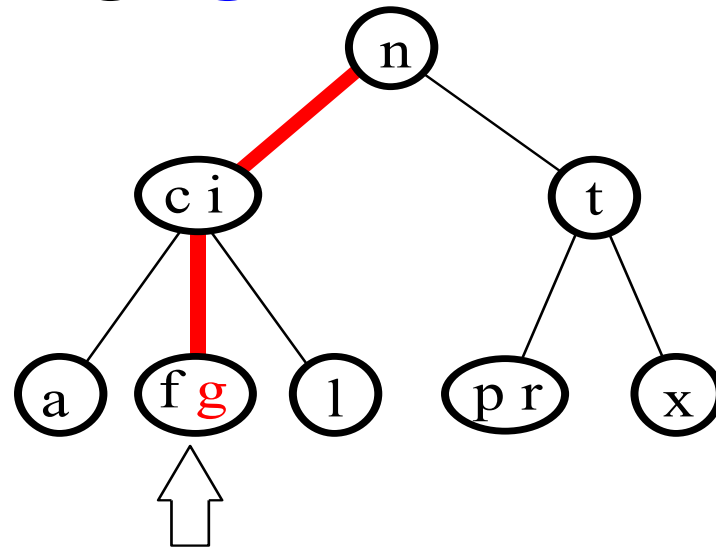
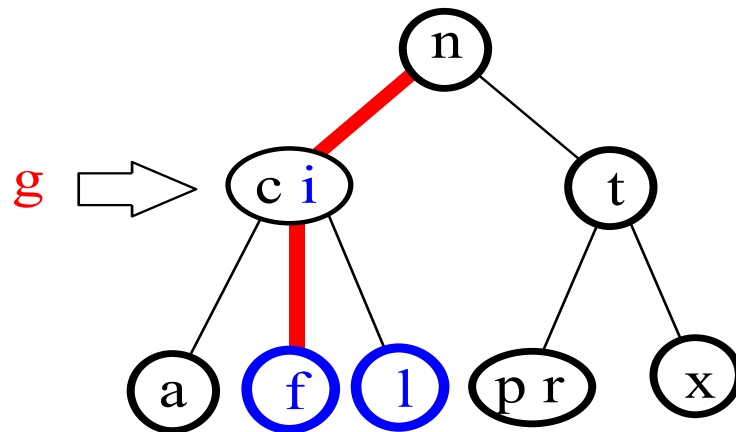
An Example



Whoa, cowboy



Whoa, cowboy



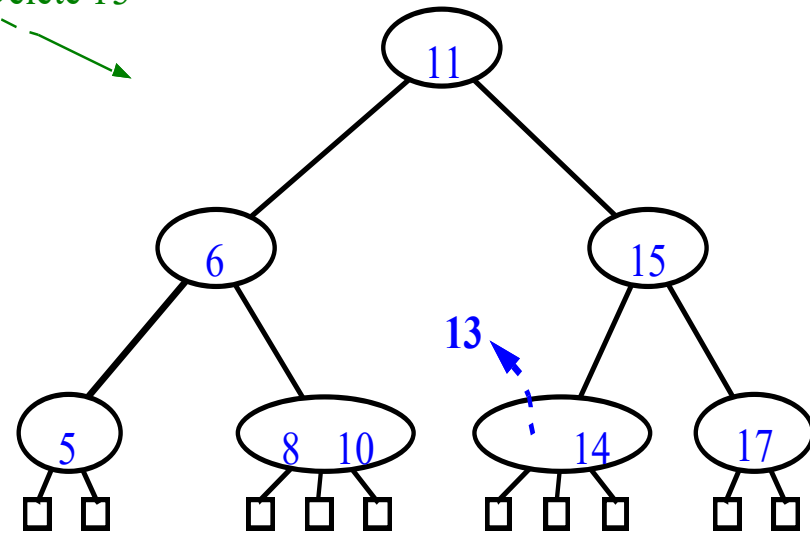
Time Complexity of Insertion in (2,4) Trees

- Time complexity:
- A search visits $O(\log N)$ nodes
- An insertion requires $O(\log N)$ node splits
- Each node split takes constant time
- Hence, operations *Search* and Insert each take time $O(\log N)$
- **Notes:**
 - Instead of doing splits top-down, we can perform them bottom-up starting at the insertion node, and only when needed. This is called *bottom-up insertion*.
 - A deletion can be performed by fusing nodes (inverse of splitting)
- **Let's take a look!**

(2,4) Deletion

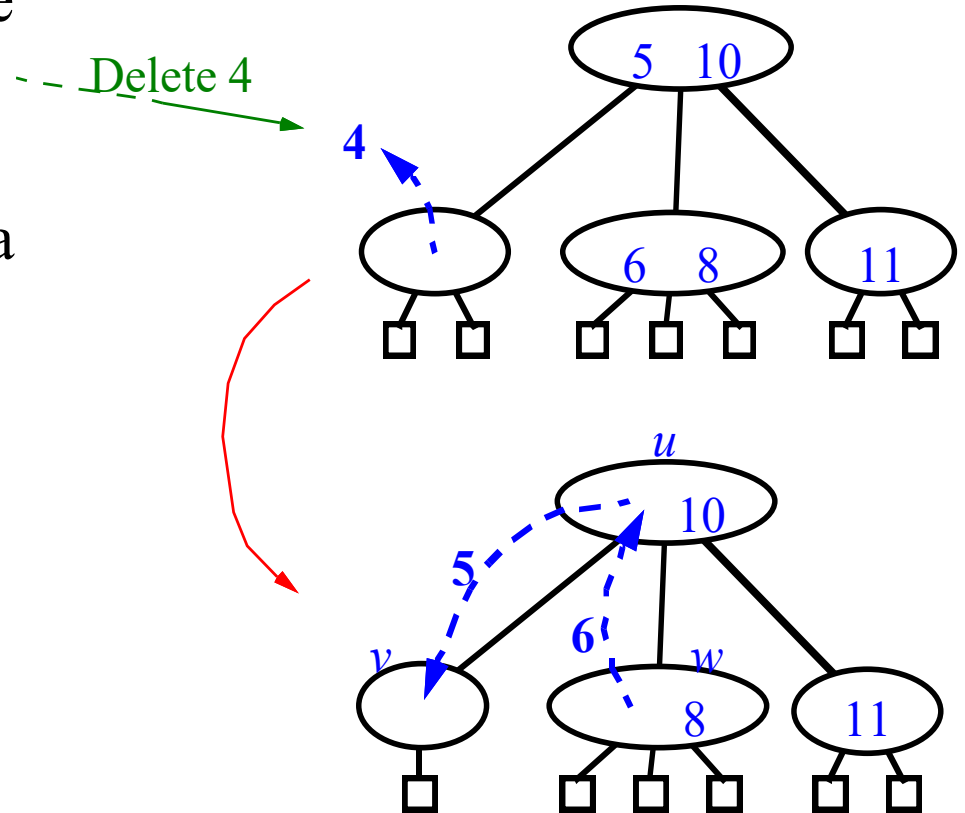
- A little trickier
- First of all, find the key with a simple multi-way search
- If the item to delete has non-external children, reduce to the case where deletable item is at the bottom of the tree:
 - Find item which precedes it in in-order traversal
 - Swap them
 - Remove the item
- Easy, right?
- ...but what about removing from 2-nodes?

Delete 13



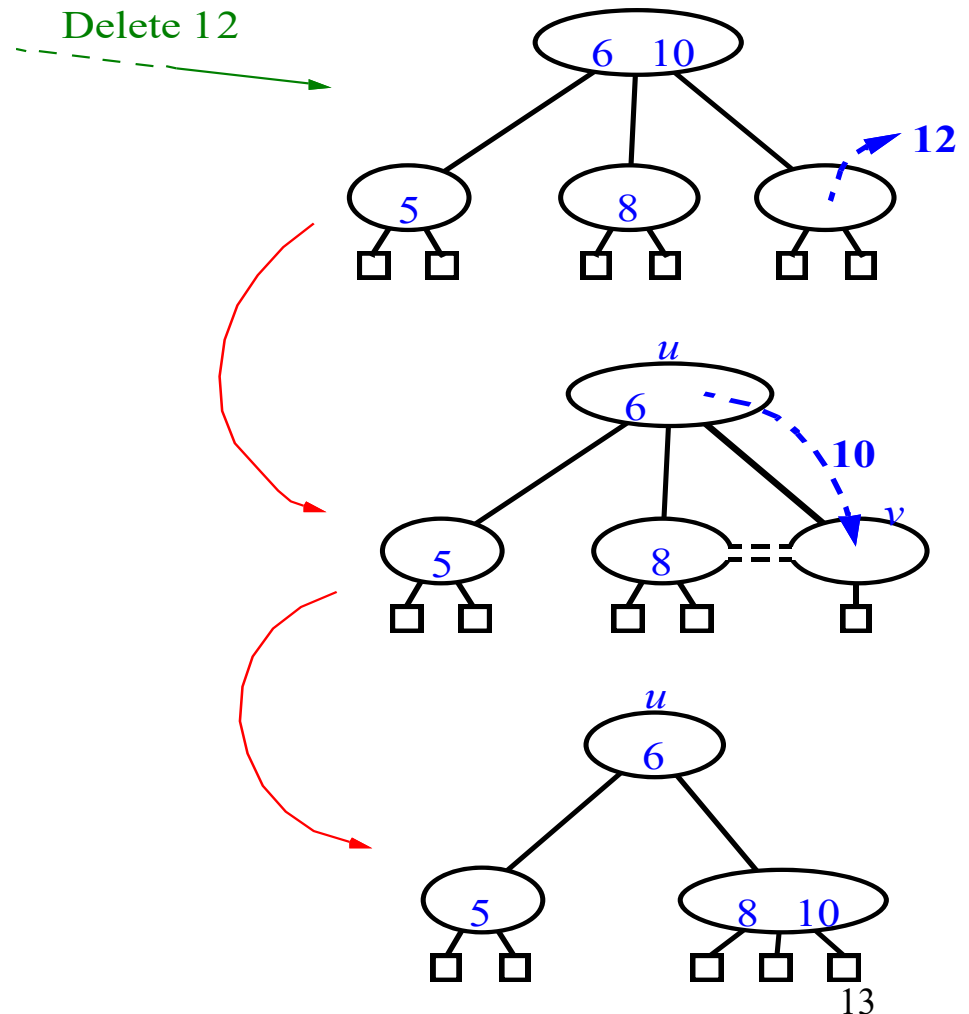
(2,4) Deletion (cont.)

- Not enough items in the node
 - *underflow*
- Pull an item from the parent, replace it with an item from a sibling
 - *called transfer*
- Still not good enough! What happens if siblings are 2-nodes?
- Could we just pull one item from the parent?
- No. Too many children
- But maybe...



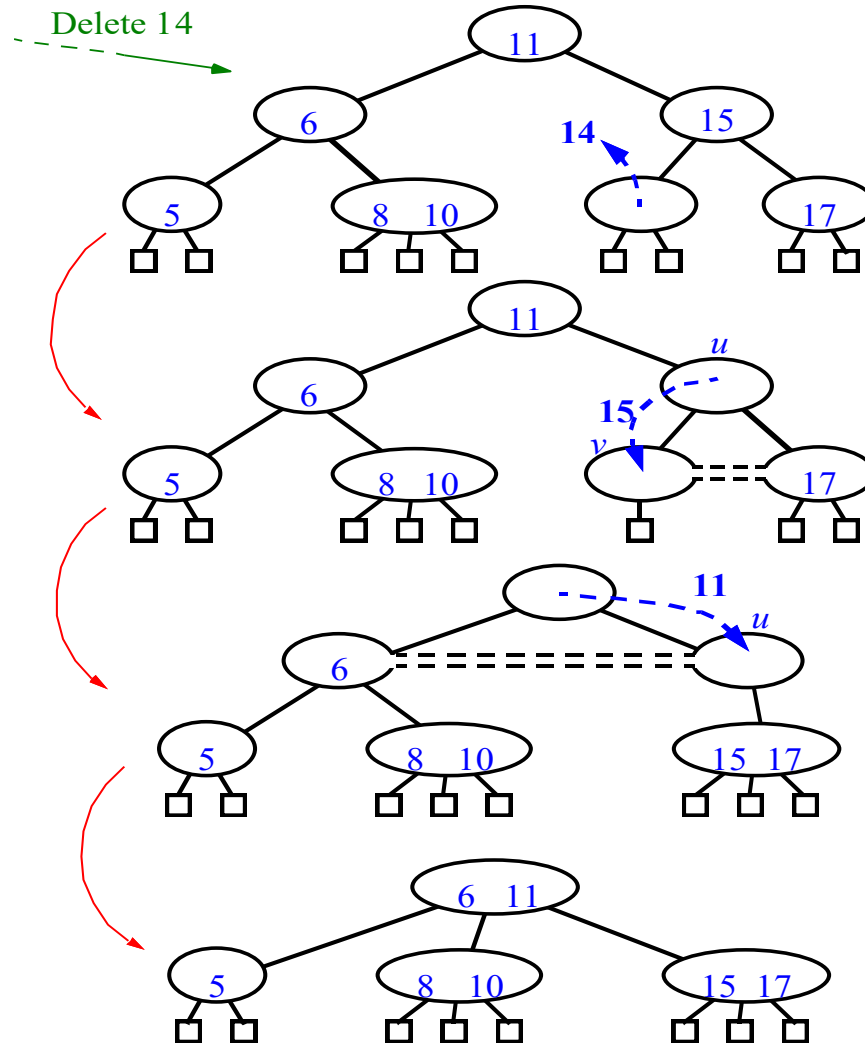
(2,4) Deletion (cont.)

- We know that the node's sibling is just a 2-node
- So we *fuse* them into one (after stealing an item from the parent, of course)
- Last special case: what if the parent was a 2-node?



(2,4) Deletion (cont.)

- Underflow can cascade up the tree, too.



(2,4) Conclusion

- The height of a (2,4) tree is $O(\log n)$.
- Split, transfer, and fusion each take $O(1)$.
- Search, insertion and deletion each take $O(\log n)$.
- Why are we doing this?
 - (2,4) trees are fun! Why else would we do it?
 - Well, there's another reason, too.
 - They're pretty fundamental to the idea of Red-Black trees as well.