# (2,4) **TREES**

- Search Trees (but not binary)
- also known as 2-4, 2-3-4 trees
- very important as basis for Red-Black trees (so pay attention!)

#### **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

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



# (2,4) Insertion

- Always maintain depth condition
- Add elements only to existing nodes



- What if that makes a node too big? *overflow*
- Must perform a *split* operation
  - replace node v with two nodes v' and v"
  - v' gets the first two keys
  - *v*" gets the last key
  - send the other key up the tree
    - if *v* is root, create new root with third key
    - otherwise just add third key to parent
- Much clearer with a few pictures...



- Tree always grows from the top, maintaining balance
- What if parent is full?



- still at most  $O(\log n)$ 

# (2,4) Deletion

- A little trickier
- First of all, find the keysimple multi-way search
- Then, 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?

# (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?
  - 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, I promise: what if the parent was a 2-node?



#### (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.
  - And you're covering Red-Black trees on Monday.
  - Perhaps more importantly, your next project is a Red-Black tree.
- Have a nice weekend!