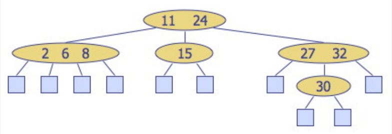
Multiway Search Trees

A *multiway search tree* allows more than one value per node.

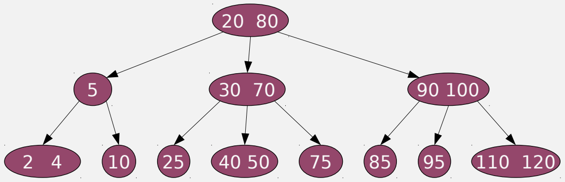
- each node has up to m-1 values, in sorted order
- a node with k values has k+1 children (which may be empty)
- *i*th subtree of a node [v_1, …, v_k] only contains values in the range $v_i \leq v < v_{i+1}$
  - $0 \leq i \leq k$
  - $v_o = -\infty$, $v_{k+1} = \infty$



2-4 Trees

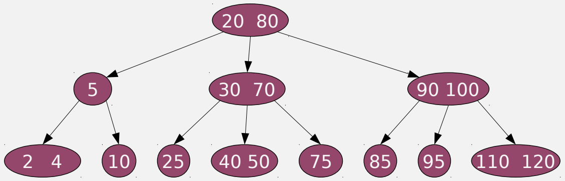
A *2-4 tree* is a multiway search tree where

- all leaves are at the same depth
- each node has 1, 2, or 3 keys and (# keys)+1 children



The height of a 2-4 tree containing n elements is O(log n).

Operations on 2-4 Trees



Searching in a multiway tree is similar to searching in a binary tree –

if the target element is not one of the keys in the current node, continue the search with the appropriate child.

Operations on 2-4 Trees

For insert and remove, we use the same approach as with AVL trees:

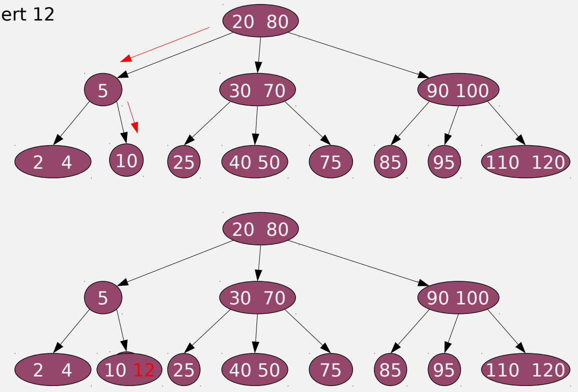
- insert/remove as dictated by the structural and ordering rules
- fix up the broken node size property as needed

Insert

New elements are always inserted at a leaf.

Insert

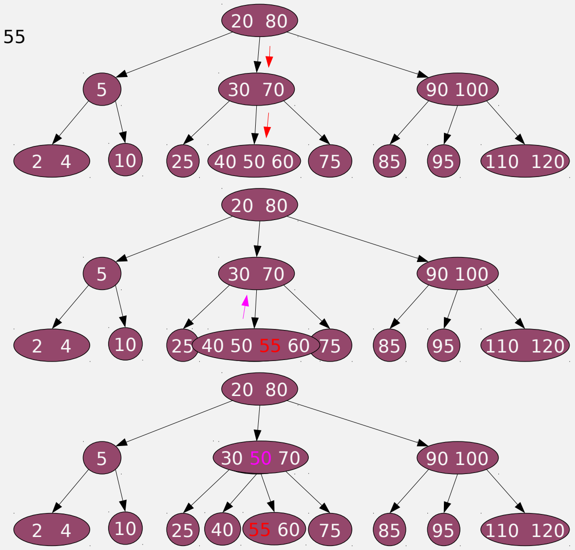
insert 12



Insert

- if insertion creates an overflow, split the node and promote a middle item to the proper place in the parent
- repeat until there are no more overflows, creating a new root if necessary

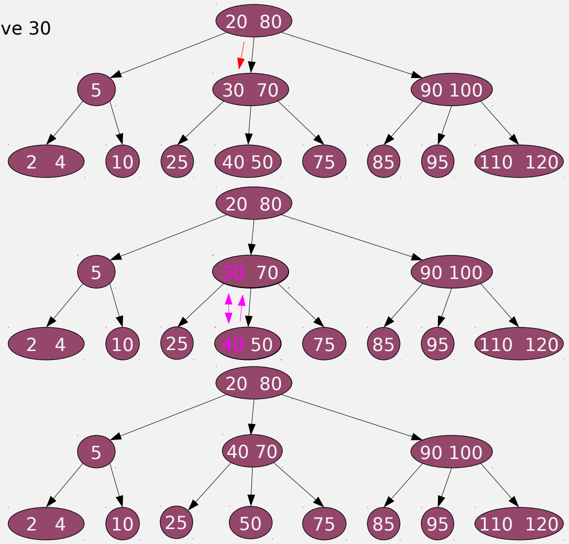
insert 55



Remove

Elements can only be removed from leaves –
first swap with next larger (or smaller) if needed.

remove 30

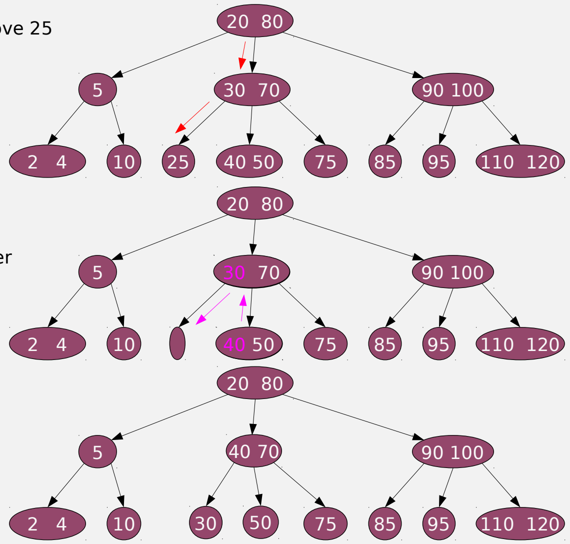


swap

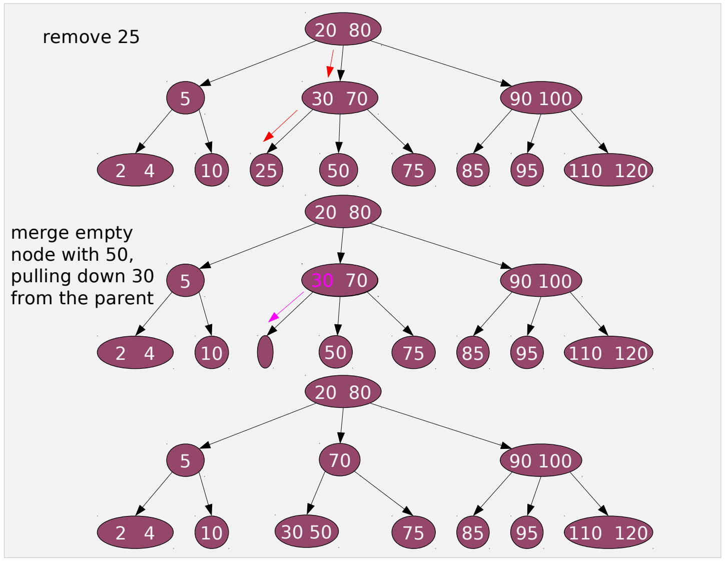
Remove

- if removal creates an underflow,
  - if there's a sibling with at least two keys, transfer one (via the parent)
  - otherwise, merge – move a key from the parent, merging the node with a sibling
- repeat until there are no more underflows, removing the root if necessary

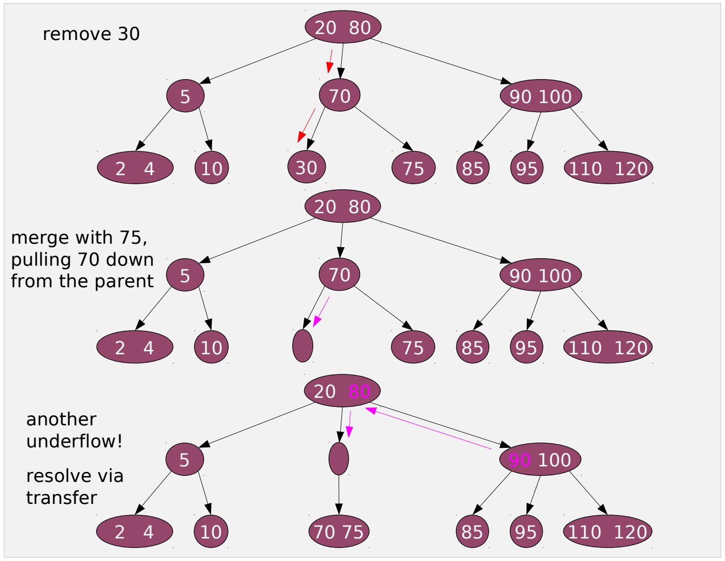
remove 25



transfer


remove 25

merge empty node with 50, pulling down 30 from the parent


remove 30

merge with 75, pulling 70 down from the parent

another underflow!

resolve via transfer

Running Time

- time for initial insert – O(log n)
- time to fix up one overflow – O(1)
- number of overflows to fix – O(log n)

→ total time for insert – O(log n)

- time for initial remove – O(log n)
- time to fix up one underflow – O(1)
- number of underflows to fix – O(log n)

→ total time for remove – O(log n)