15.3 2-3-4 Trees & Other Trees

2-3-4 Trees
extend BST to have more than 2 children
need to have different relational check than just less-than & greater-than
will allow more than 2 search paths
m-node tree has m children
stores values k1 to k(m-1)
children T1 to Tm
check value v as follows:
- $v < k_1$ go to $T_1$
- $k_1 \leq v < k_2$ go to $T_2$
- $k_2 \leq v < k_3$ go to $T_3$
- etc until:
- $k(m-1) \leq v$ go to $T_m$

2-3-4 tree allows $m = 2, 3$ or $4$
BST is $m = 2$ only

2-3-4 ADT
Data: A tree where
1. each node stores 1 to 3 values
2. each non-leaf node is an m-node w/ $m = 2, 3$ or $4$
3. all leaves are on the same level
Operations
create empty
check empty
search for an item
insert an item; maintain 2-3-4 property
delete an item; maintain 2-3-4 property

Example:

```
       5 3
      /   \
  27, 38 /   \
 /     /   \
16, 25 33, 36 41, 46, 48
```

Inserting an item
Must maintain property 3 which keeps the tree balanced
Pseudocode
if tree is empty
    create 2-node w/ item & make it root
else
    find the leaf node where item belongs
    if leaf contains < 3 valves
        add item to leaf
    else
        split leaf into two nodes
        median of 4 values used as "root" for this subtree
        all values < median go into one node (1 or 2 values)
        all values > median go into the other (1 or 2 values)
        set node to original leaf
        set parent to node
        set split to true
while split is true
    if parent is NULL
        create new 2-node w/ median
        make two new nodes children of 2-node
        set root to 2-node
    else if parent has < 3 values
        add median to parent values
        replace node w/ two new nodes
        set split to false
    else
        split parent into two nodes using same method as above
        set node to parent
        set parent to parent's parent
Example:
An alternative to splitting up w/ the while loop is to split all 4-node to two 2-nodes while searching for leaf to insert the item. This is called top-down insertion. It eliminates the while loop and is faster since only visit each node once.

Data Storage
Simple implementation
array of 3 for values
array of 4 node pointers for children
Simple implementation is inefficient
always allocates space for a 4-node
wasted memory for 2-node & 3-node
approximately 75% of memory is wasted
can use BST to represent any tree but BST will not stay balanced
red-black trees can also be used to represent 2-3-4 trees

Red-Black trees
BST tree w/ colored links (red & black)
kept balanced using AVL-like rotations
maintains the following properties:
1. Each path from the root to a leaf node has the same number of black links
2. No path from the root to a leaf has two or more consecutive red links
Note: this is one definition. An alternative definition is:
each node is colored red or black
the root node is black
if a node is red, its children must be black
every path from a node to a NULL "leaf" must contain the same number of black nodes
To represent 2-3-4 tree as red-black
make the link black if it is an actual link in the 2-3-4 tree
make the link red if it connects parts of the same node in 2-3-4 tree

splitting a node will change link colors
the split may cause two consecutive red links
for example, there could be a red link to Y
use AVL rotations to remove consecutive red links

B-Trees
book's definition is weak, using another
use m-node concept like 2-3-4 tree
can use external storage for data
data items are stored in leaves, which can be on disk
Definition
all data is stored in leaves
non-leaf nodes store keys to data on disk
the root is either a leaf or has between 2 and M children
the non-leaf nodes have ceiling(M/2) to M children
all leaves have ceiling(L/2) to L data items
choose L & M based on amount of data to be stored
affects number of nodes needed to index the data
nodes are in main memory, so want to choose values that will allow all nodes to be stored
L & M can be the same
2-3-4 tree has L=M= 4
Example: