CSE 332: Data Structures and Parallelism

Section 4: Balanced Trees Solutions

0. The ABC’s of AVL Trees

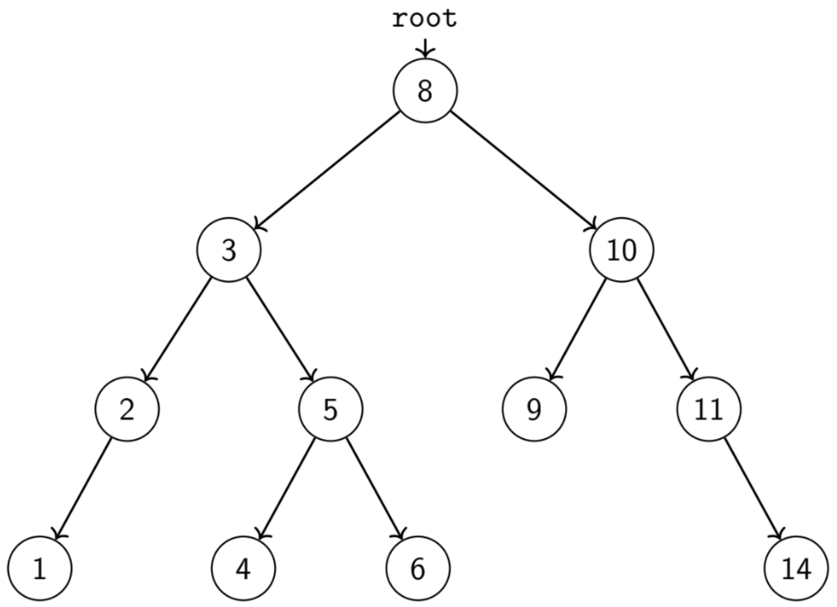
What are the constraints on the data types you can store in an AVL tree? When is an AVL tree preferred over another dictionary implementation, such as a HashMap?

AVL trees are similar to TreeMaps. The constraint is that they require that keys be comparable. The value type can be anything, just like any other dictionary.

A perk over HashMaps is that keys can be iterated over in sorted order. AVL trees are also preferred over BSTs when there’s a possibility of sorted input because the balancing prevents the worst case of a degenerate tree.

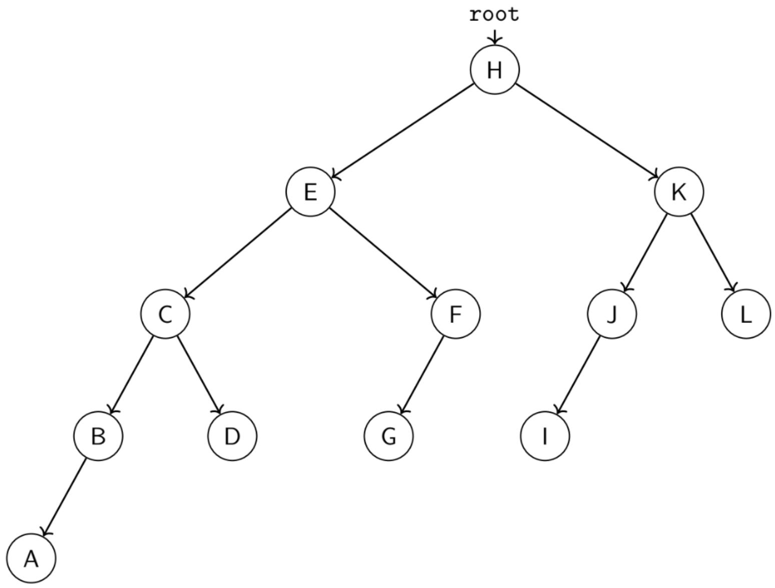
1. Let’s Plant an AVL Tree

Insert `10, 4, 5, 8, 9, 6, 11, 3, 2, 1, 14` into an initially empty AVL Tree.



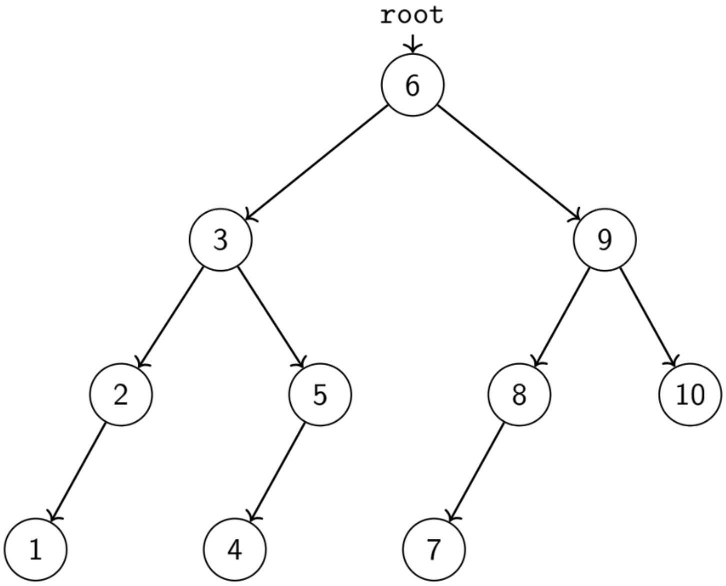
2. MinVL Trees

Draw an AVL tree of height 4 that contains the minimum possible number of nodes.



3. AVL Trees

Insert `6, 5, 4, 3, 2, 1, 10, 9, 8, 7` into an initially empty AVL Tree.



4. The ABC's of B-Trees

a) What properties must a B-tree of n values have with given values for M and L?

- B-Tree order property:
  - Every subtree between keys a and b contains all data x where $a \leq x < b$
  - The values in the leaves are in key sorted order
  - The keys in the internal nodes are stored in sorted order
- B-Tree structure property:
  - If $n \leq L$, the root is a leaf with n values, otherwise the root is an internal node that must have between 2 and M children
  - All internal nodes must have between $ceil\left(\frac{M}{2}\right)$ and M children (i.e., half-full)
  - All leaf nodes must have between $ceil\left(\frac{L}{2}\right)$ and L key-value pairs (i.e., half-full)
  - All leaf nodes must be at the same depth

b) Give an example of a situation that would be a good job for a B-tree. Furthermore, are there any constraints on the data that B-trees can store?

B-trees are most appropriate for very, very large data stores, like databases, where the majority of the data lives on disk and cannot possibly fit into RAM all at once.

B-trees require orderable keys. B-trees are typically not implemented in Java because what makes them worthwhile is their precise management of memory.

5. Implement a B-Tree? Nah, Let’s Analyze!

Given the following parameters for a B-Tree with a page size of 256 bytes:

- Key Size = 8 bytes
- Pointer Size = 2 bytes
- Data Size = 14 bytes per record (includes the key)

Assuming that M and L were chosen appropriately, what are M and L? Recall that M is defined as the maximum number of pointers in an internal node, and L is defined as the maximum number of values in a leaf node. Give a numeric answer and a short justification based on two equations using the parameter values above.

We start by defining the following variables:

- 1 page on disk is b bytes
- Keys are k bytes
- Pointers are t bytes
- Key/Value pairs are v bytes

We know that the amount of memory used by one leaf node is vL and the amount of memory used by one internal node is $tM + k(M - 1)$. We want select values for M and L such that both equations are $\leq b$.

If we solve both equations for M and L, we obtain

$M = floor\left(\frac{b+k}{t+k}\right)$ and $L = floor\left(\frac{b}{v}\right)$

Plugging in the given values, we get

$M = floor\left(\frac{256+8}{2+8}\right) = 26$ and $L = floor\left(\frac{256}{14}\right) = 18$

6. Oh, B-Trees

Find a tight upper bound on the worst case runtime of these operations on a B-tree. Your answers should be in terms of M, L, and n.

a) Looking up the value of a key

a) We must do a binary search on a node containing M pointers, which takes $\mathcal{O}(\lg(M))$ time, once at each level of the tree.
b) There are $\mathcal{O}\left(\log_M(n)\right)$ levels.
c) We must do a binary search on a leaf of L elements, which takes $\mathcal{O}(\lg(L))$
d) Putting it all together, a tight bound on the runtime is $\mathcal{O}\left(\lg(M)\log_M(n) + \lg(L)\right)$.

b) Inserting and deleting a key-value pair

The steps for insert and delete are similar and have the same worst case runtime.

a) Find the leaf: $\mathcal{O}\left(\lg(M)\log_M(n)\right)$. Same runtime as *looking up the value of a key*.
b) Insert/remove in the leaf – there are L elements, essentially stored in an array: $\mathcal{O}(L)$
c) Split a leaf/merge neighbors: $\mathcal{O}(L)$
d) Split/merge parents, in the worst case going up to the root: $\mathcal{O}\left(M\log_M(n)\right)$

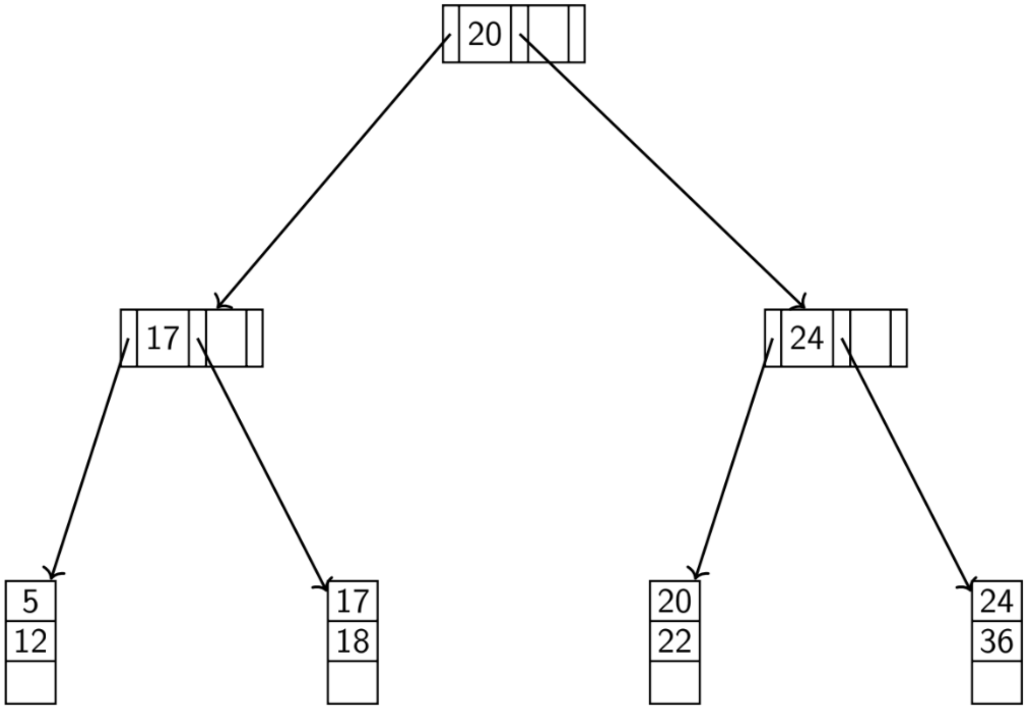
The total cost is then $\lg(M)\log_M(n) + 2L + M\log_M(n)$

We can simplify this to a worst-case runtime $\mathcal{O}\left(L + M\log_M(n)\right)$ by combining constants and observing that $M\log_M(n)$ dominates $\lg(M)\log_M(n)$. Note that in the average case, splits for any reasonably-sized B-tree are rare, so we can amortize the work of splitting over many operations.

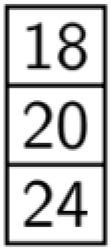
However, if we're using a B-tree, it's because what concerns us the most is the penalty of disk accesses. In that case, we might find it more useful to look at the worst-case number of disk lookup operations in the B-tree, which is $\mathcal{O}\left(\log_M(n)\right)$.

7. B-Trees

a) Insert the following into an empty B-Tree with $M = 3$ and $L = 3$:
`12, 24, 36, 17, 18, 5, 22, 20`
(Assume these numbers are both the keys and the values.)



b) Delete `17, 12, 22, 5, 36`
(Assume these numbers are both the keys and the values.)



c) Given the following parameters for a B-Tree with $M = 11$ and $L = 8$

- Key Size = 10 bytes
- Pointer Size = 2 bytes
- Data Size = 16 bytes per record (includes the key)

Assuming that M and L were chosen appropriately, what is the likely page size on the machine where this implementation will be deployed? Give a numeric answer and a short justification based on two equations using the parameter values above.

We start by defining the following variables:

- 1 page on disk is b bytes
- Keys are k bytes
- Pointers are t bytes
- Key/Value pairs are v bytes

We know that the amount of memory used by one leaf node is vL and the amount of memory used by one internal node is $tM + k(M - 1)$. We want select values for M and L such that both equations are $\leq b$.

If we solve both equations for M and L, we obtain

$M = floor\left(\frac{b+k}{t+k}\right)$ and $L = floor\left(\frac{b}{v}\right)$

Plugging in the given values, we get

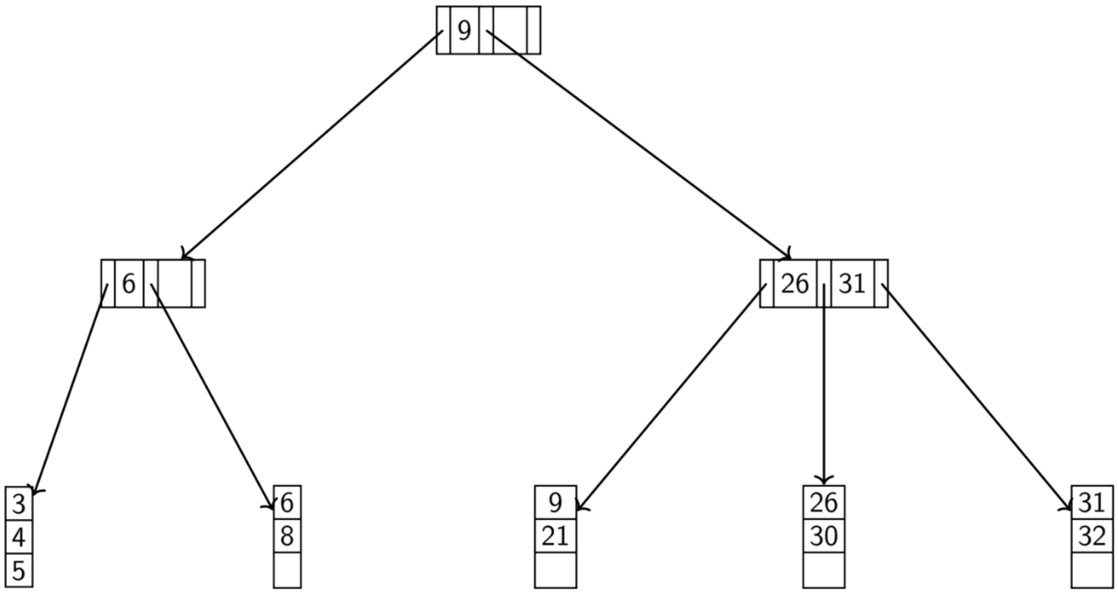
$M = floor\left(\frac{b+10}{2+10}\right)$ and $L = floor\left(\frac{b}{16}\right)$

And solving for b gives us:

$b = 128$ bytes

8. It's Fun to B-Trees!

a) Insert the following into an empty B-Tree with $M = 3$ and $L = 3$:
3, 32, 9, 26, 6, 21, 8, 4, 5, 30, 31
(Assume these numbers are both the keys and the values.)



b) Delete 4, 5, 21, 9, 31, 3, 26, 8
(Assume these numbers are both the keys and the values.)