B-Trees

Prof. Chris Jermaine
cmj4@cs.rice.edu

Now, Back To Algorithms (no Java!)

- One very common linked structure is a “B-Tree”
- It is a very fast way to implement a map:
  - — $O(\lg(n))$ finds
  - — $O(\lg(n))$ inserts
  - — $O(\lg(n) + m)$ “range” finds
- Since nodes can be arbitrarily large (n-ary, not binary tree)
  - — B-trees were originally used as file-based structures
  - — Each node was the size of a disk block
- But now, B-trees are arguably faster than BSTs in RAM, too
  - — Since BSTs are binary, they often don’t fill up a cache line
  - — A B-tree with node size close to cache line size is very, very fast

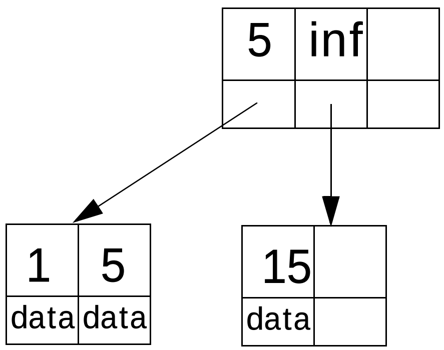
B-Trees

- Have two node types:
  - — "Internal" nodes
  - — "Leaf" nodes
- Internal nodes
  - — Store a list of (at most) $n_{internal}$ (key, ptr) pairs
  - — Here, "ptr" or "pointer" might be a Java reference, or a file name and byte offset, or an IP address plus a process ID plus a memory address, or...
  - — "ptr" refers to another B-tree whose root can be found at that location
- Leaf nodes
  - — Store a list of (at most) n_{leaf} (key, data) pairs
  - — Note difference: no data in internal nodes, just keys!

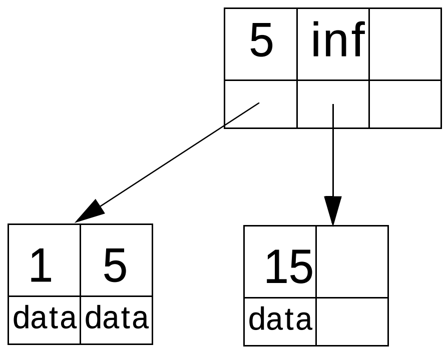
B-Tree Invariants

- The tree is totally “height balanced”
  - — Every “pointer” in a B-Tree node
  - — Refers to a tree of exactly the same height
  - — So every path from root to leaf in tree is same length
- The tree is ordered
  - — Consider the (key_i, ptr_i) pair at position i in an internal node
  - — Every data item in the tree referred to by ptr_j (for $j <= i$) must have a key <= key_i
- The tree is at least half full
  - — Every internal node has at least ($n_{internal}$ / 2) pairs
  - — Every leaf node has at least (n_{leaf} / 2) pairs

Example B-Tree

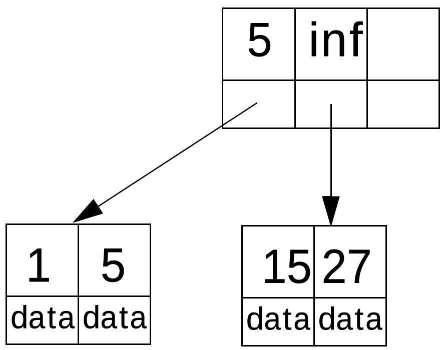


Example B-Tree

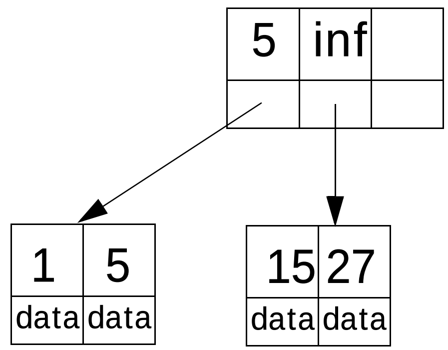


- Say we want to add a (27, data) pair...

Example B-Tree

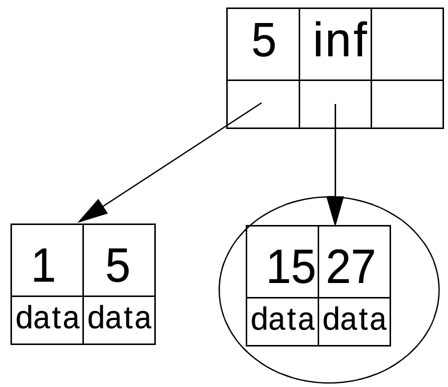


Example B-Tree



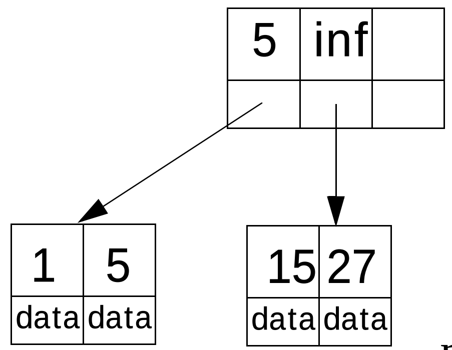
- Say we want to add a (16, data) pair...
-

Example B-Tree



- Say we want to add a (16, data) pair...
  - — Oops! The appropriate leaf node is already full

Example B-Tree



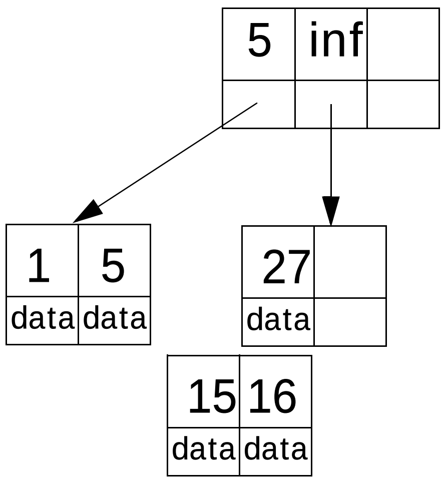
15, 16, 27

median always goes on LHS

in case of even number, median is large in the lower half

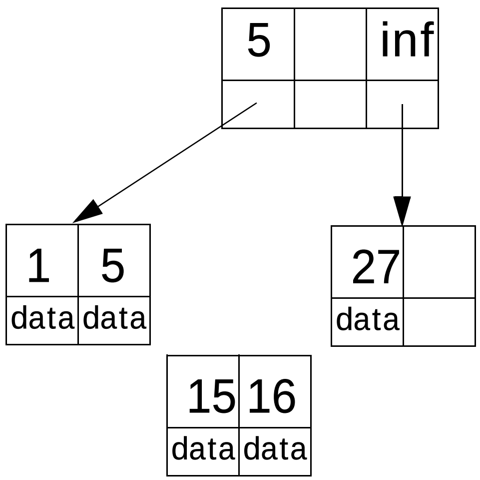
- So... we perform a leaf node “split”
  - — Step 1: sort all pairs using the keys, and partition via the median

Example B-Tree



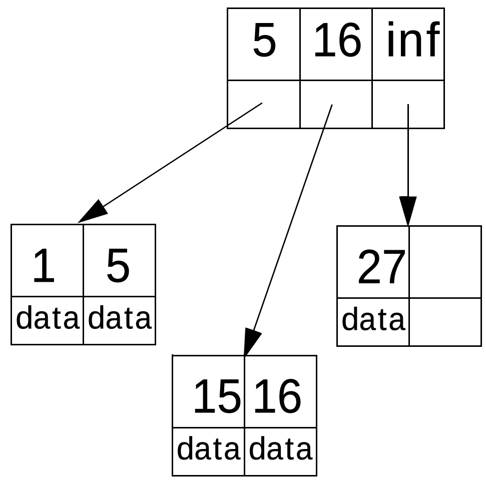
- So...
  - — Step 1: sort all pairs using the keys, and partition via the median
  - — Step 2: put lower half into new leaf node

Example B-Tree



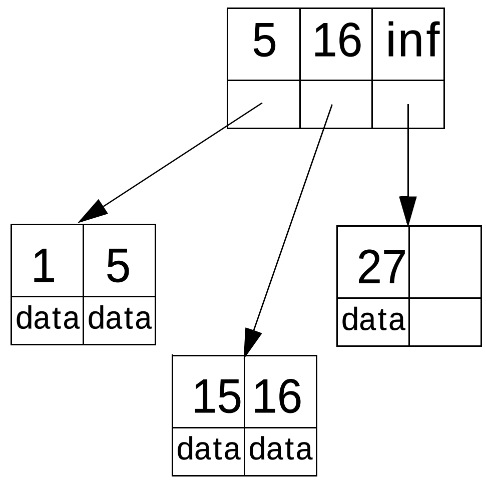
- So...
  - — Step 1: sort all pairs using the keys, and partition via the median
  - — Step 2: put lower half into new leaf node
  - — Step 3: slide (key, pointer) pairs in parent over one slot to make room for new pair

Example B-Tree



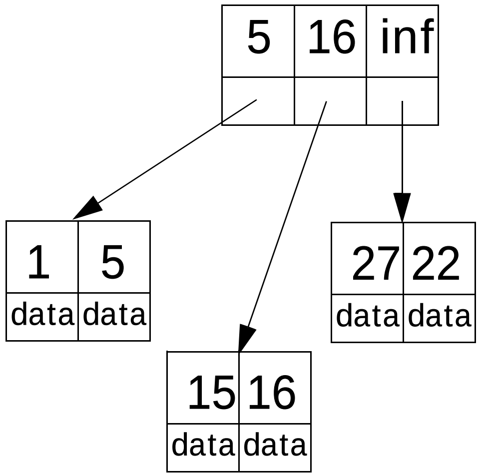
- So...
  - — Step 1: sort all pairs using the keys, and partition via the median
  - — Step 2: put lower half into new leaf node
  - — Step 3: slide (key, pointer) pairs in parent over one slot to make room for new pair
  - — Step 4: add the pair (median, ptr to new node) to the parent... DONE!

Example B-Tree

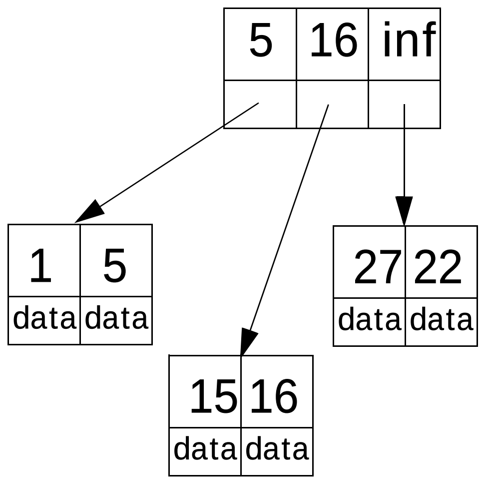


- Now we add a (22, data) pair... easy!

Example B-Tree

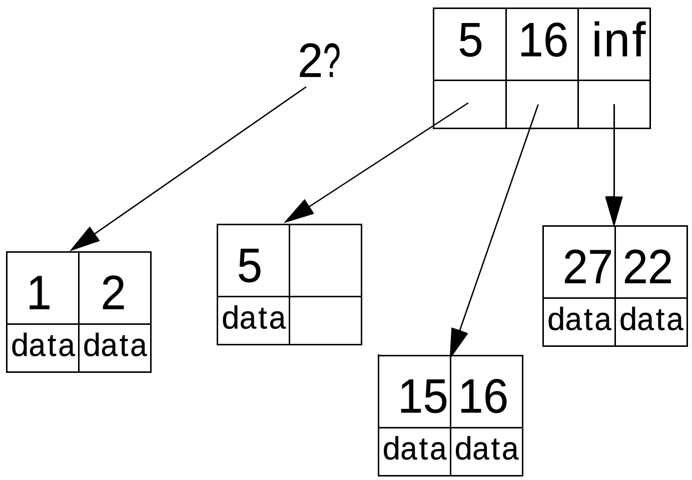


Example B-Tree



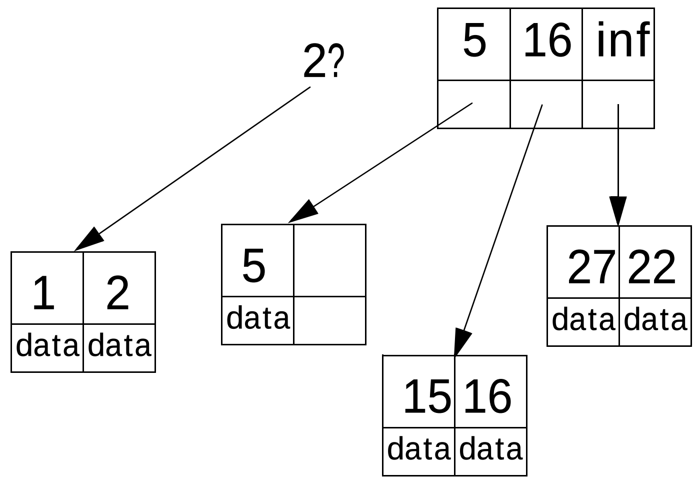
- What happens when a (2, data) pair is added?

Example B-Tree



- What happens when a (2, data) pair is added?
  - Same steps as before, except that we can't slide everything in parent over

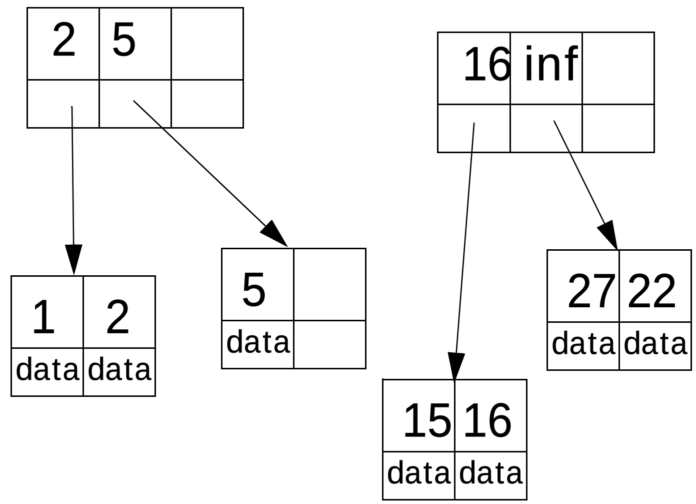
Example B-Tree



2, 5, | 16, inf

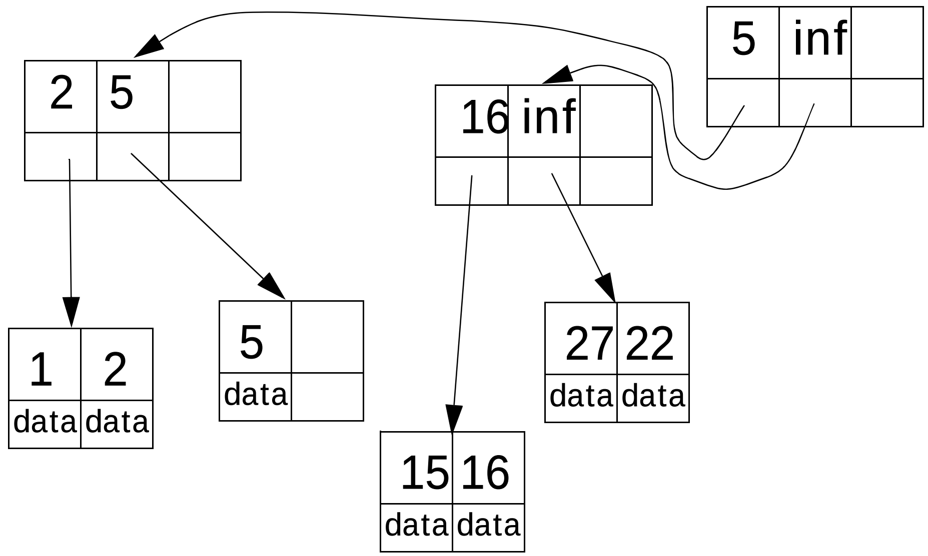
- So we need to split the internal node
  - Step 1: sort and partition via the median

Example B-Tree



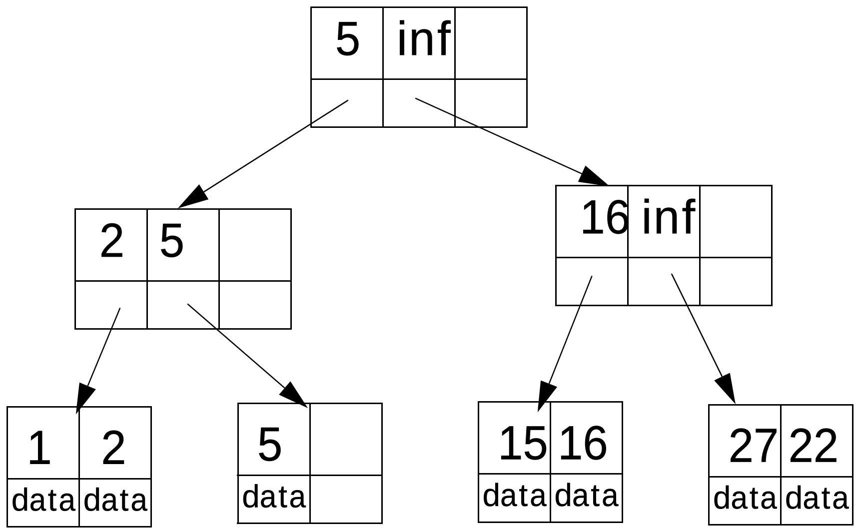
- So we need to split the internal node
  - — Step 1: sort and partition via the median
  - — Step 2: put lower half into a new internal node

Example B-Tree



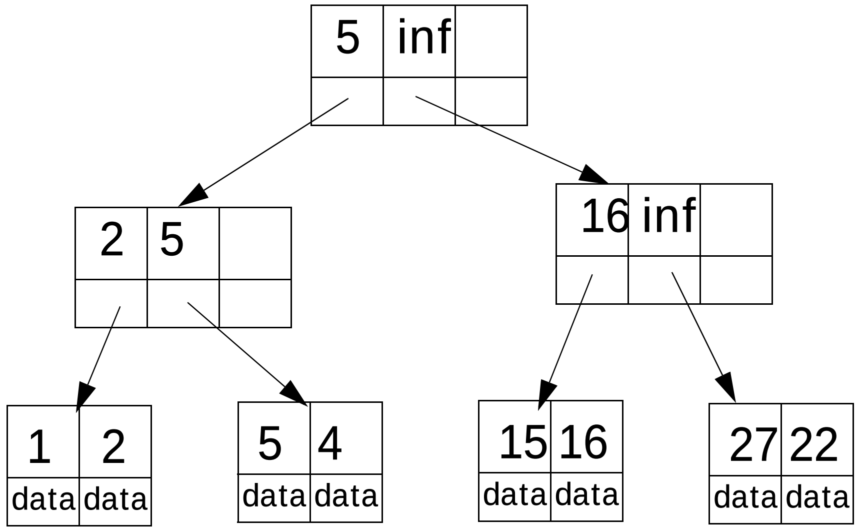
- So we need to split the internal node
  - Step 1: sort and partition via the median
  - Step 2: put lower half into a new internal node
  - Step 3: since we split the root, create a new root w. two (key, ptr) pairs... first pair is (median, ptr to new node)... second pair is (inf, ptr to split node) DONE!

Example B-Tree



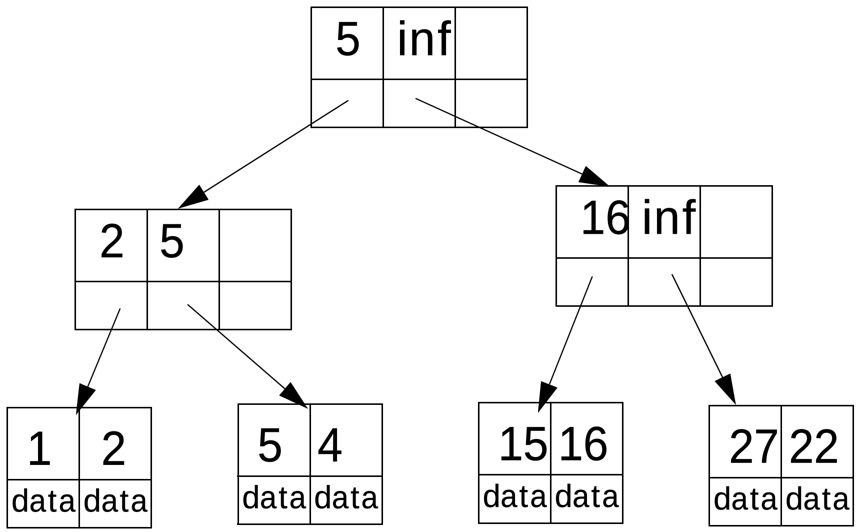
- Let’s make this tree look a little nicer...

Example B-Tree



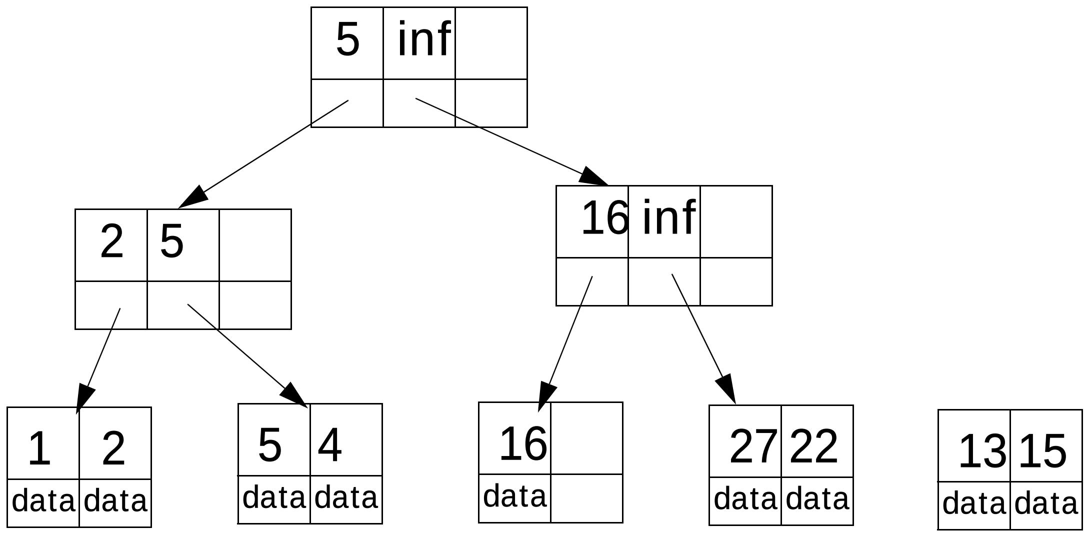
- Let’s make this tree look a little nicer...
  - — and then add a (4, data) pair

Example B-Tree



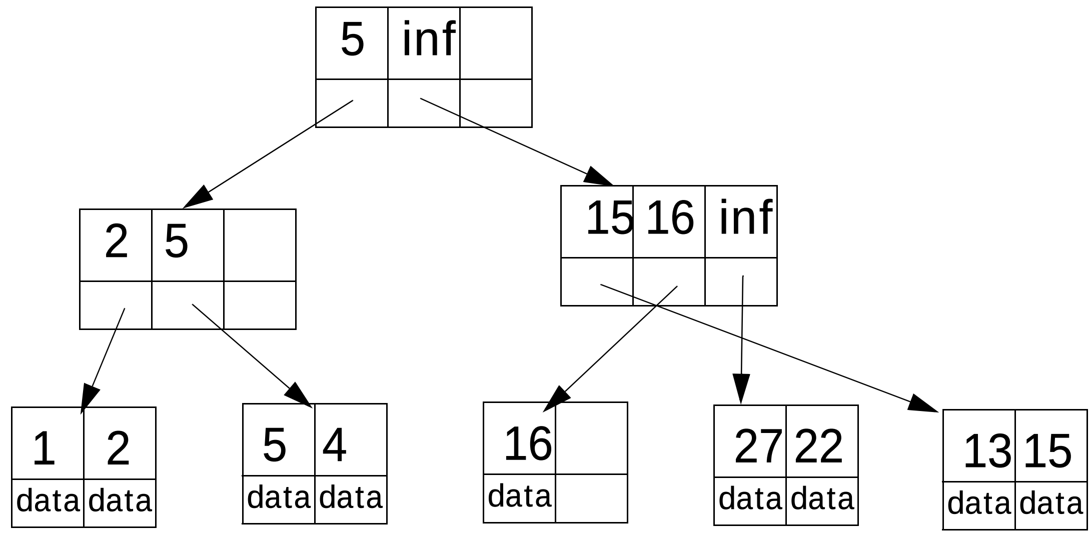
- Let’s make this tree look a little nicer...
  - — and then add a (4, data) pair
  - — and then a (13, data) pair, which causes a split...

Example B-Tree



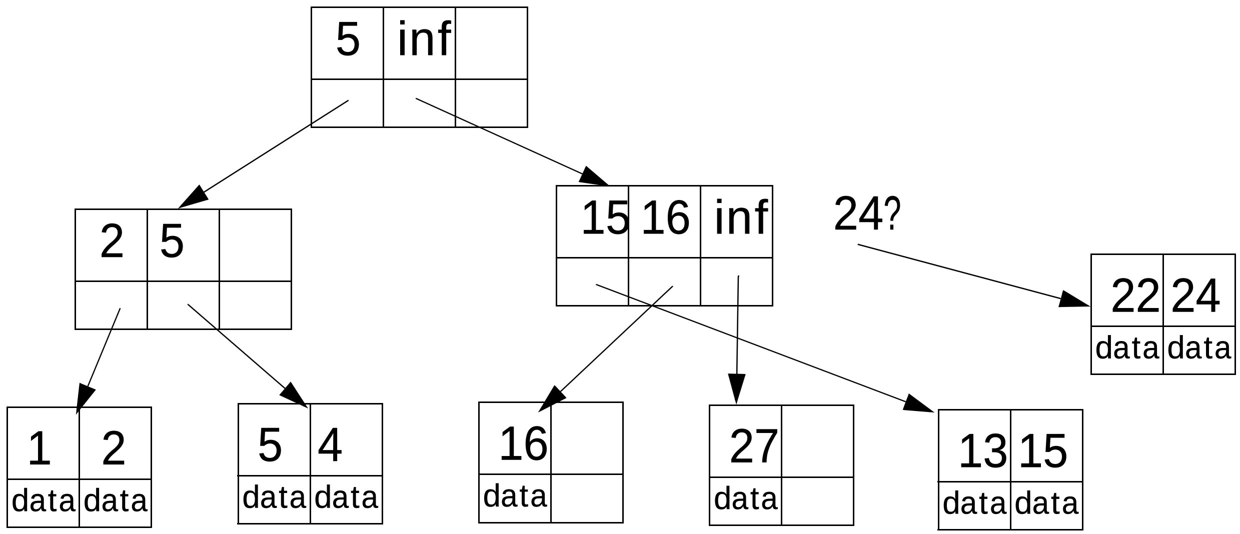
- Let's make this tree look a little nicer...
  - — and then add a (4, data) pair
  - — and then a (13, data) pair, which causes a split...

Example B-Tree



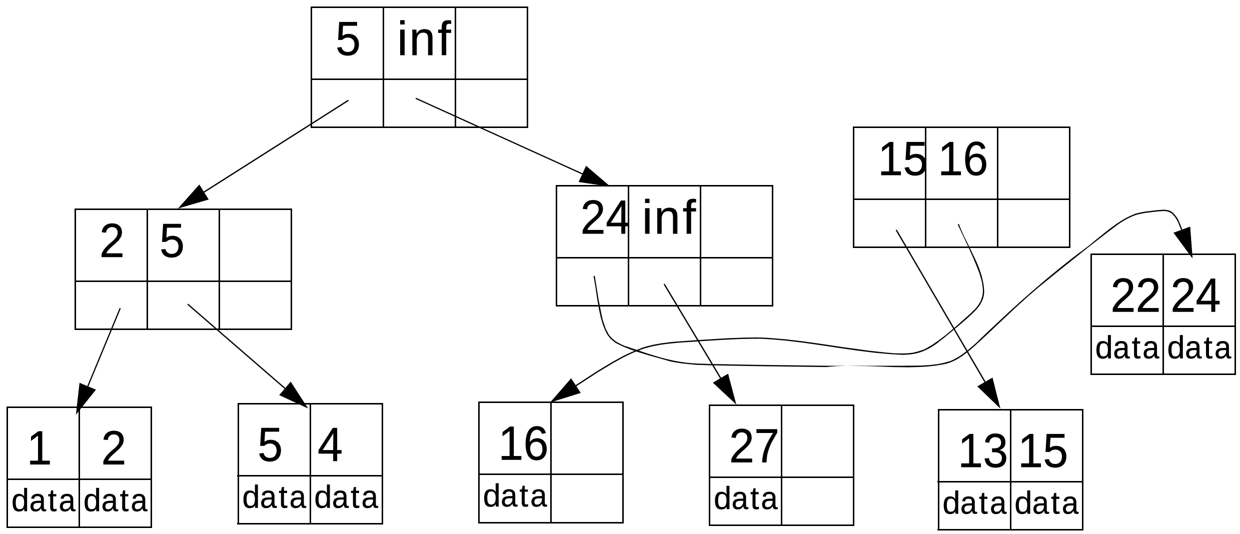
- Let's make this tree look a little nicer...
  - — and then add a (4, data) pair
  - — and then a (13, data) pair, which causes a split... and an addition to the parent

Example B-Tree



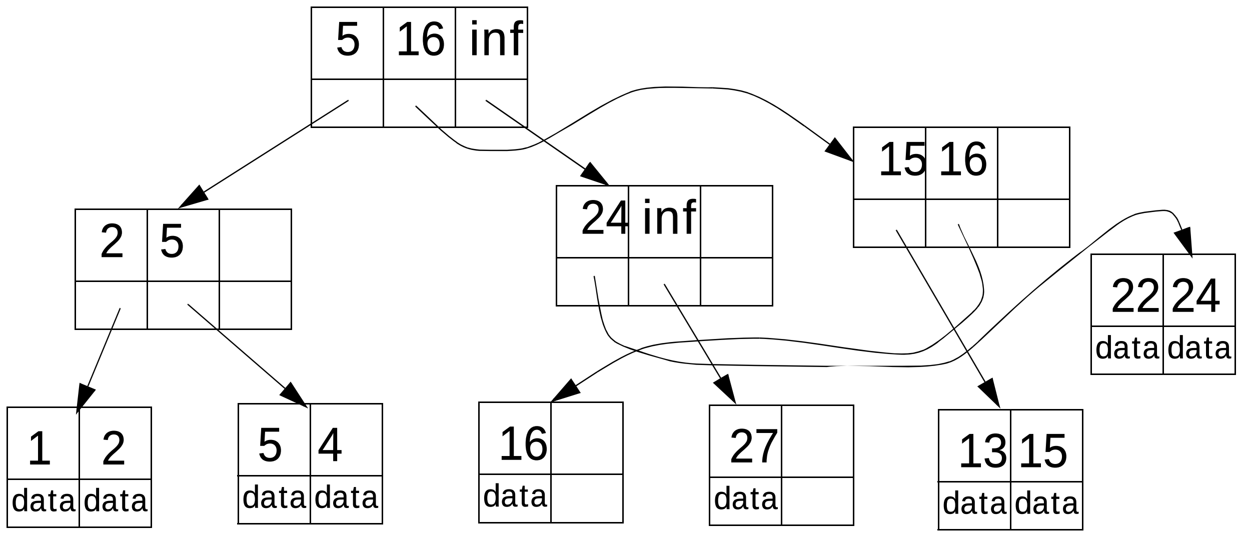
- Finally, add a (24, data) pair
  - this causes a split at the leaf

Example B-Tree



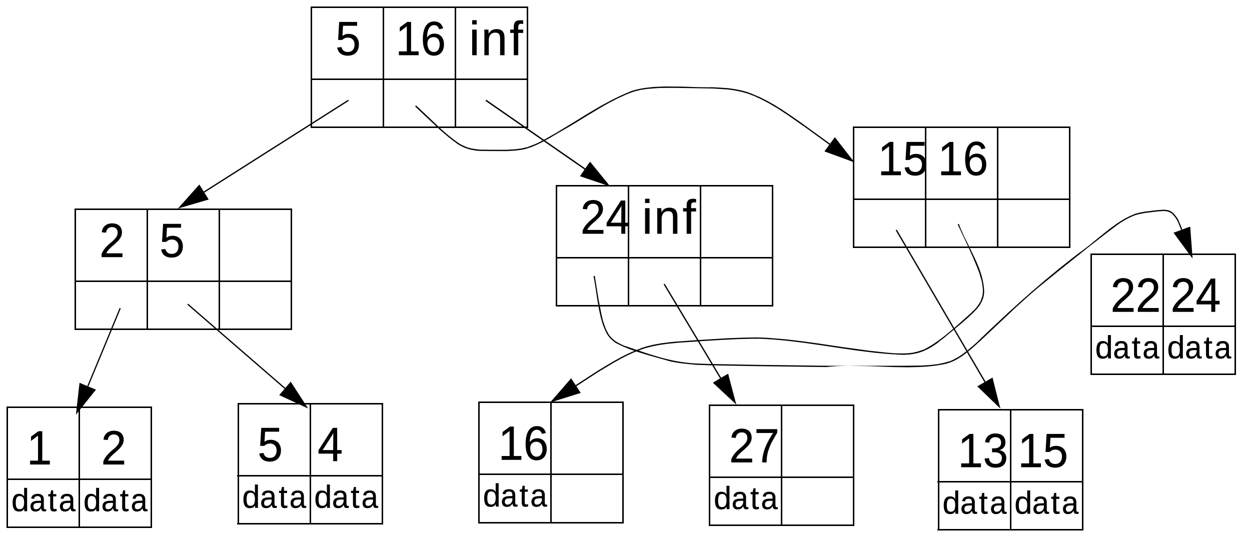
- Finally, add a (24, data) pair
  - — this causes a split at the leaf
  - — which in turn causes a split at the parent

Example B-Tree



- Finally, add a (24, data) pair
  - — this causes a split at the leaf
  - — which in turn causes a split at the parent
  - — which in turn causes an insert into the parent's parent

Example B-Tree



- Here's a worthwhile exercise to do on your own:
  - What would happen if we then added a (3, data) pair, then a (0, data) pair?

Some Final Issues

- How to do point finds?
  - — Recursively search child trees whose range could possibly intersect query point
  - — Note: if we allow repeated key vals, need to go both directions when query key appears in an internal node!
- How to do range finds?
  - — Recursively search child trees whose range could possibly intersect query range
- How to do deletes?
  - — Just go to leaf with (key, data) pair you want to delete and remove it
  - — Can “collapse” nodes if under-full, but long ago people decided this is a bad idea

Questions?