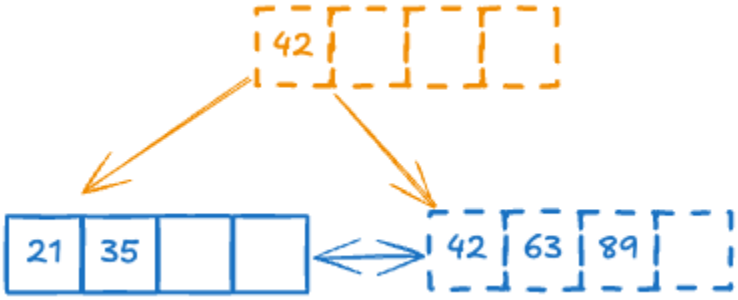
Example B+ Tree: m = 4

Step 1 - Insert: 42, 21, 63, 89

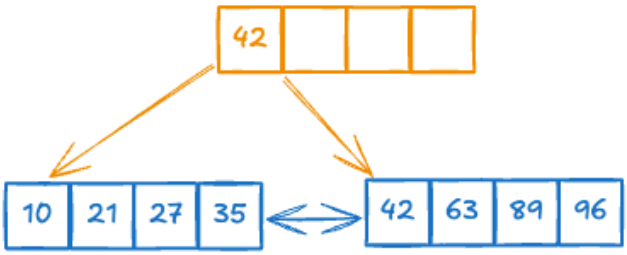
- Initially, the first node is a leaf node AND root node.
- 21, 42, … represent keys of some set of K:V pairs
- Leaf nodes store keys and data, although data not shown
- Inserting another key will cause the node to split.

Step 2 - Insert: 35



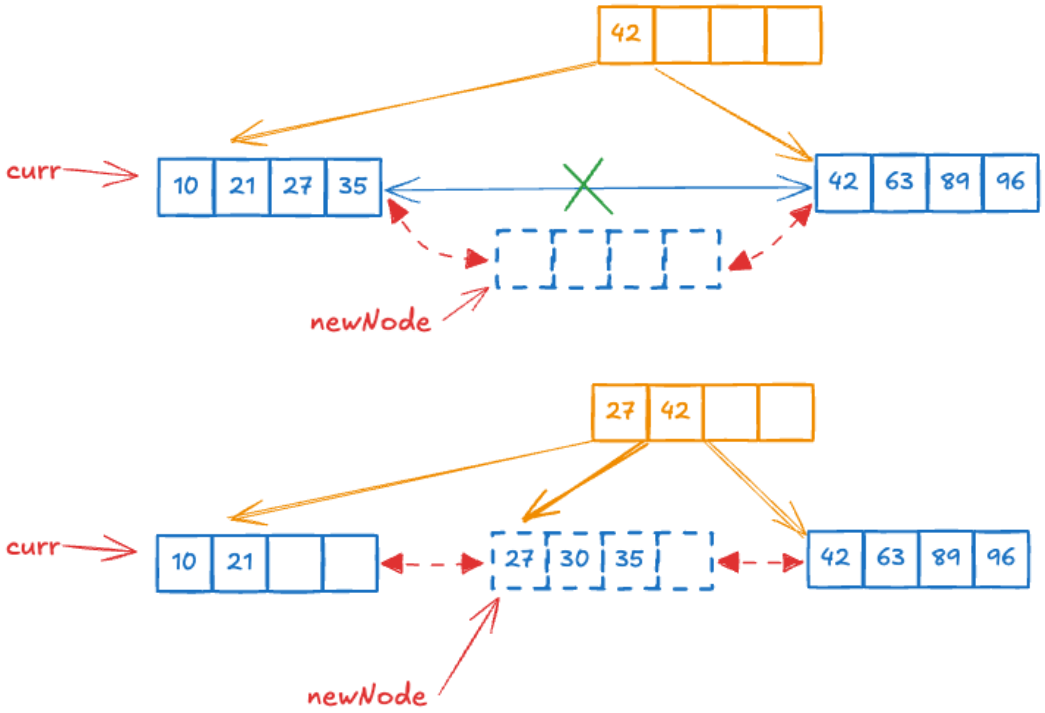
- Leaf node needs to split to accommodate 35. New leaf node allocated to the right of existing node
- 5/2 values stay in original node; remaining values moved to new node
- Smallest value from new leaf node (42) is copied up to the parent, which needs to be created in this case. It will be an internal node.

Step 3 - Insert: 10, 27, 96

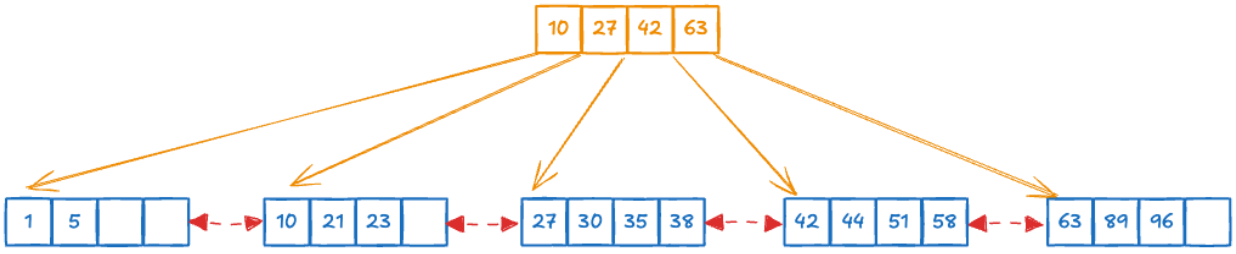


- The insert process starts at the root node. The keys of the root node are searched to find out which child node we need to descend to.
- EX: 10. Since 10 < 42, we follow the pointer to the left of 42
- Note - none of these new values cause a node to split

Step 4 - Insert 30



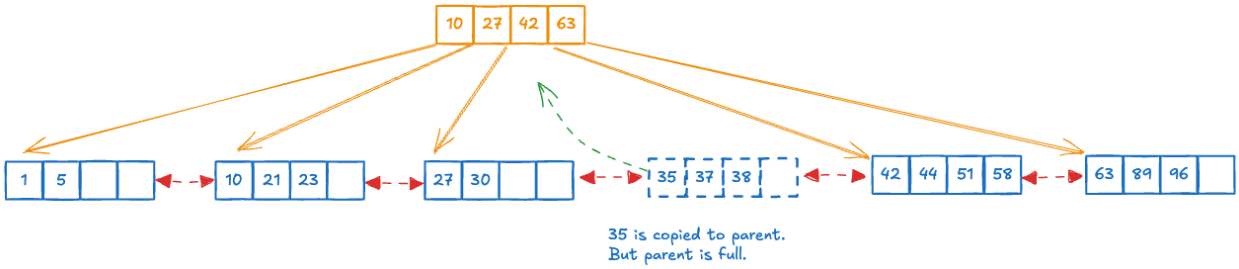
- Starting at root, we descend to the left-most child (we'll call curr).
  - curr is a leaf node. Thus, we insert 30 into curr.
  - BUT curr is full. So we have to split.
  - Create a new node to the right of curr, temporarily called newNode.
  - Insert newNode into the doubly linked list of leaf nodes.
- re-distribute the keys
- copy the smallest key (27 in this case) from newNode to parent; rearrange keys and pointers in parent node.
- Parent of newNode is also root. So, nothing else to do



Observation: The root node is full.

- The next insertion that splits a leaf will cause the root to split, and thus the tree will get 1 level deeper.

Step 4 - Insert 37



- When splitting an internal node, we move the middle element to the parent (instead of copying it).
- In this particular tree, that means we have to create a new internal node which is also now the root.