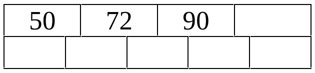
The course is winding down to the end. This week is the last tutorial where there will be any instruction. Next week, I'll just ask you to go to the tutorial and answer students' questions about assignment #5 and also questions about any other material in the course (in preparation for the final exam).

So for this week, all I want you to do is to walk through some examples of adding/deleting keys to/from B-trees. To (re)familiarize yourself with B-trees you can look at my course notes (sections 11: http://www.site.uottawa.ca/ftppub/courses/Winter/csi2131/sec11.html).

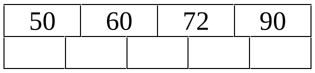
Example 1: B-tree of order 5

We start the exercise with a B-tree of order M = 5. Every node (except the parent) in a B-tree of order 5 must have at least 2 values ((M-1)/2) and at most 4 values (M-1).

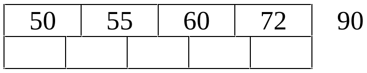
At the beginning, the B-tree has three values: 50, 72 and 90 all in the root.



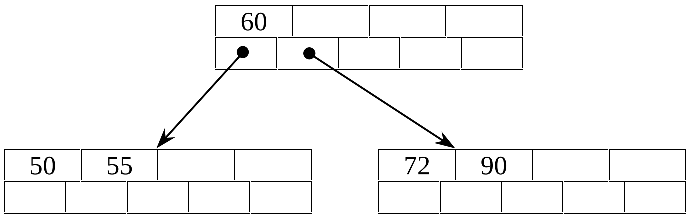
1. ***Add the value 60 to the B-tree:*** There is room for 60 in the root, so we just add it in the correct position (maintaining the order of values in the node).



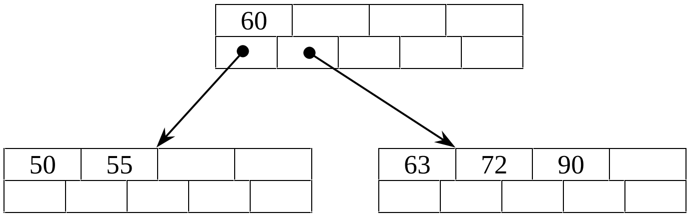
2. ***Add the value 55 to the B-tree:*** The root node is full, so adding 55 causes it to overflow.



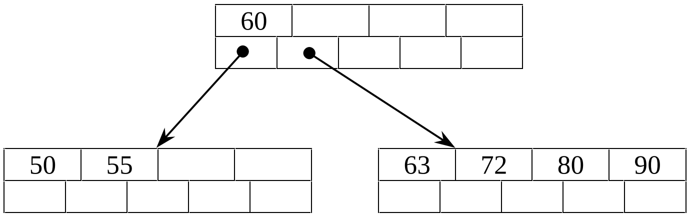
   Repair the overflow by splitting the node into three parts: 50 55 | 60 | 72 90. The left part becomes a new node, the right part becomes a new node, and the middle part gets promoted to the parent of the two new nodes.



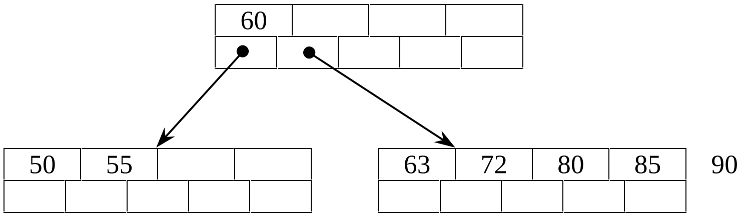
3. ***Add 63 to the B-tree:*** 63 is greater than 60, so it must go somewhere in 60's right subtree. There is only one node in the subtree, so 60 should go in that leaf. There is room left in the leaf, so no overflow occurs.



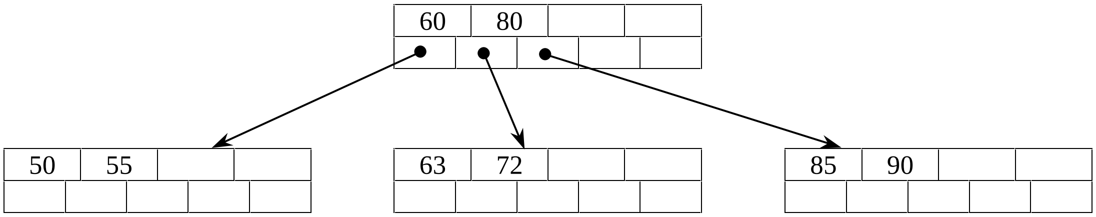
4. ***Add 80:*** 80 is greater than 60 so it belongs in 60's right subtree. There is still room, so no overflow occurs.



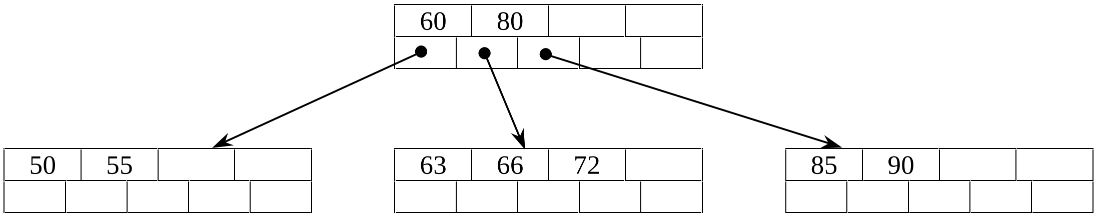
5. ***Add 85:*** 85 is greater than 60 so put it in 60's right child. The right child overflows.



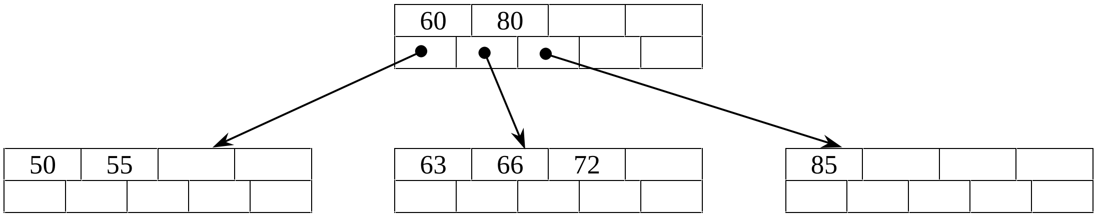
Repair the overflow by splitting the node into three parts: 63 72 | 80 | 85 90. The left part becomes a new node, the right part becomes a new node, and the middle part gets promoted to the parent of the two new nodes.



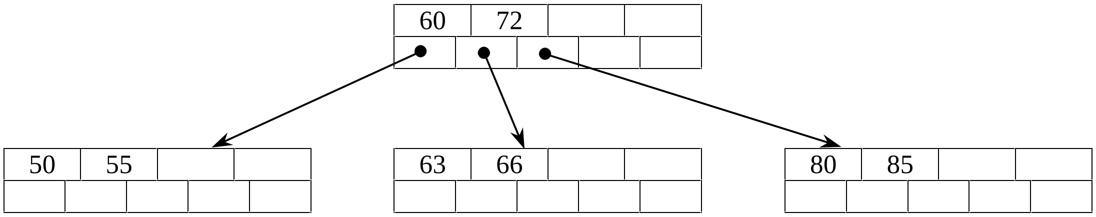
6. ***Add 66:*** 66 is greater than 60 and less than 80, so it belongs in the middle child. There is room in the middle child, so simply add 66 in the correct position.



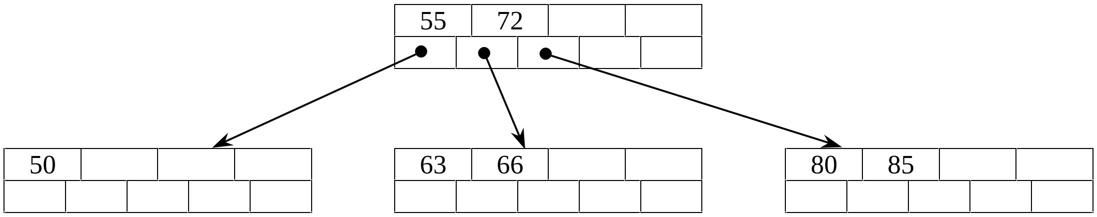
7. ***Delete 90:*** We are only allowed to delete from leaves. 90 is already in a leaf, so we can go ahead and delete it.



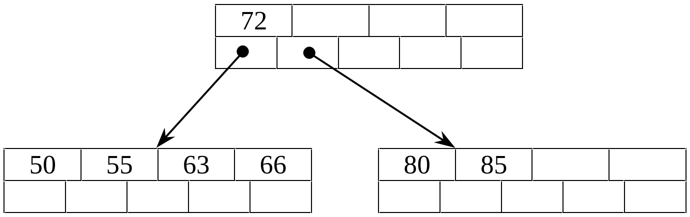
Deletion causes underflow (remember, every node must have at least two values in it). To repair the underflow, we combine the underflowing node with its more populous neighbour (actually, the underflowing node here has only one neighbour) and their parent key. There are more than M-1 values in this combination, so we split the combination into left, middle and right: 63 66 | 72 | 80 85. The left part and right part become new nodes with the middle part as their parent.



8. ***Delete 60:*** We're only allowed to delete from a leaf. 60 is not in a leaf, so we must first swap it with the biggest value in its left subtree. 60 is then guaranteed to be in a leaf so we can delete it.

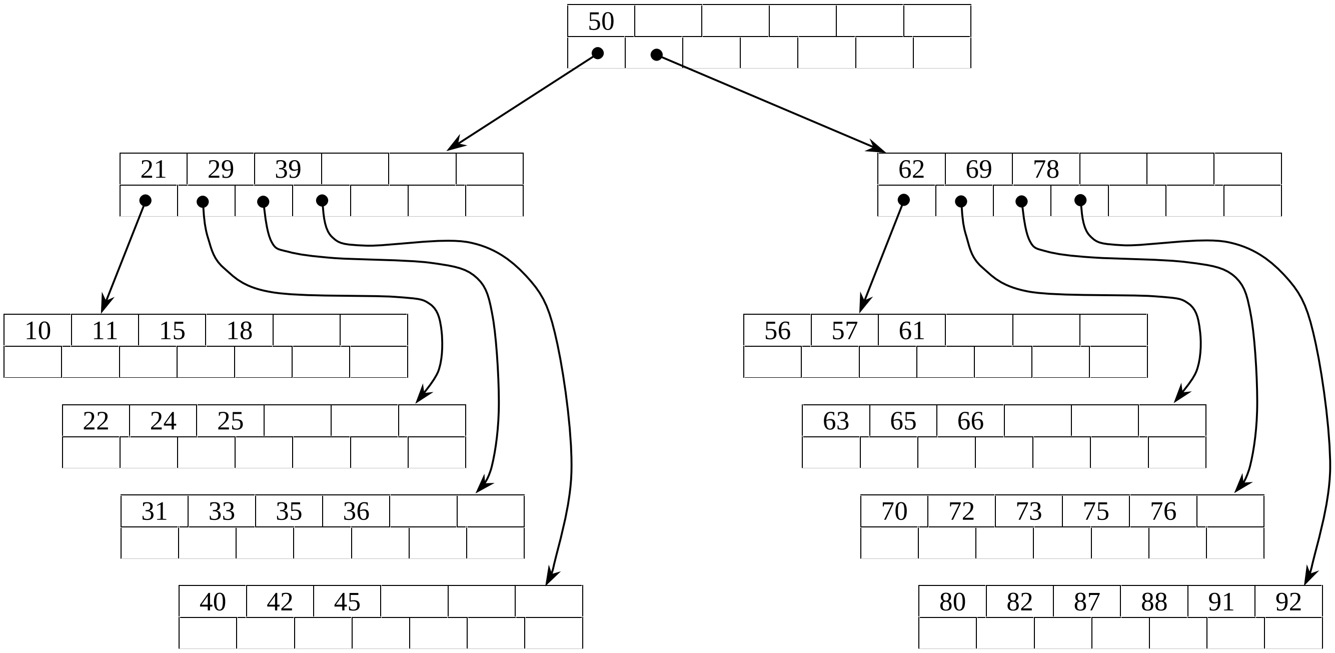


Deletion causes underflow in the left leaf. We repair the underflow by combining the underflowing node with its more populous neighbour and their parent: 50 55 63 66. There are only M-1 values in the combination, so we don't split it. It becomes a single node replacing the previous two nodes.

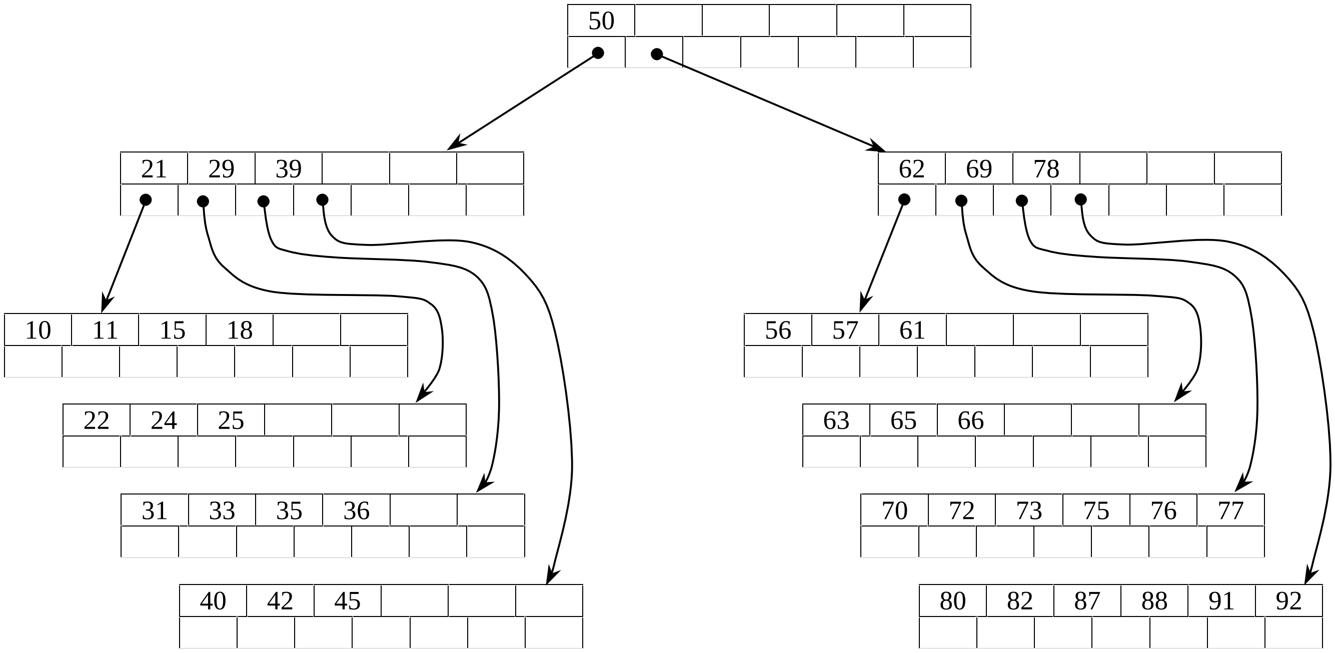


Example 2: B-tree of order 7

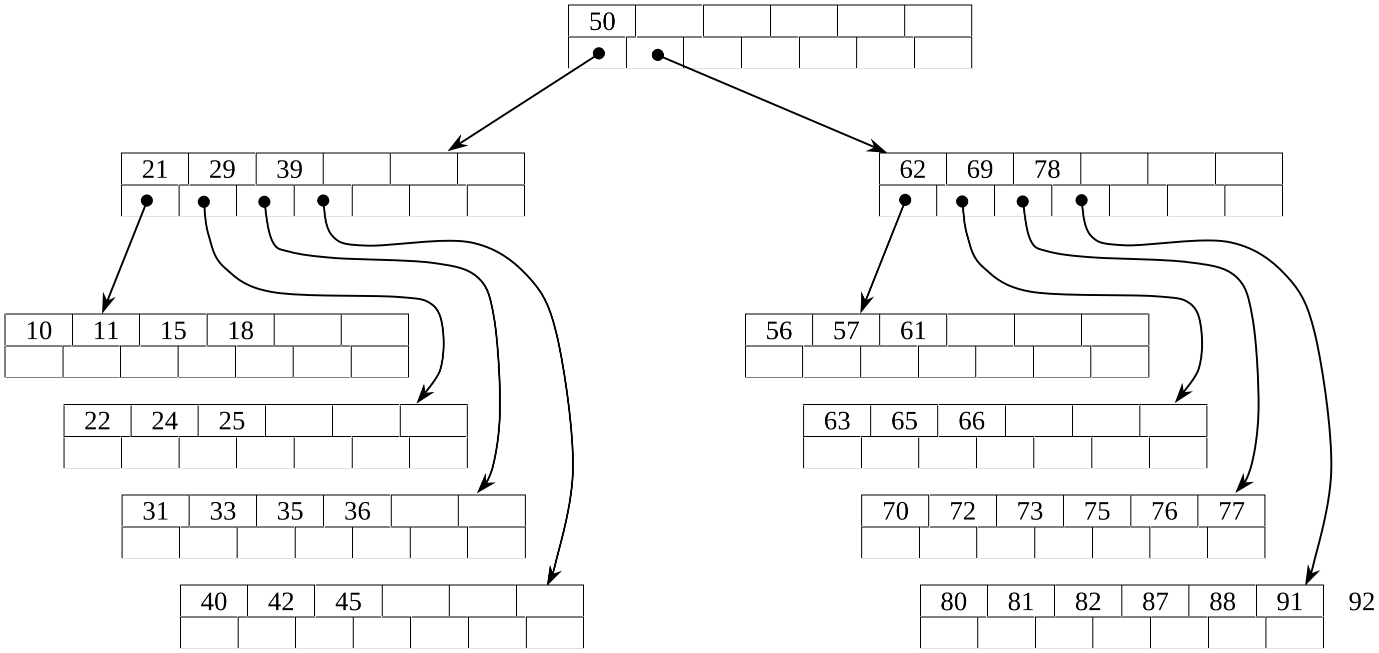
Let’s have a look at a slightly bigger example: a B-tree of order 7 with lots of nodes already in the tree.



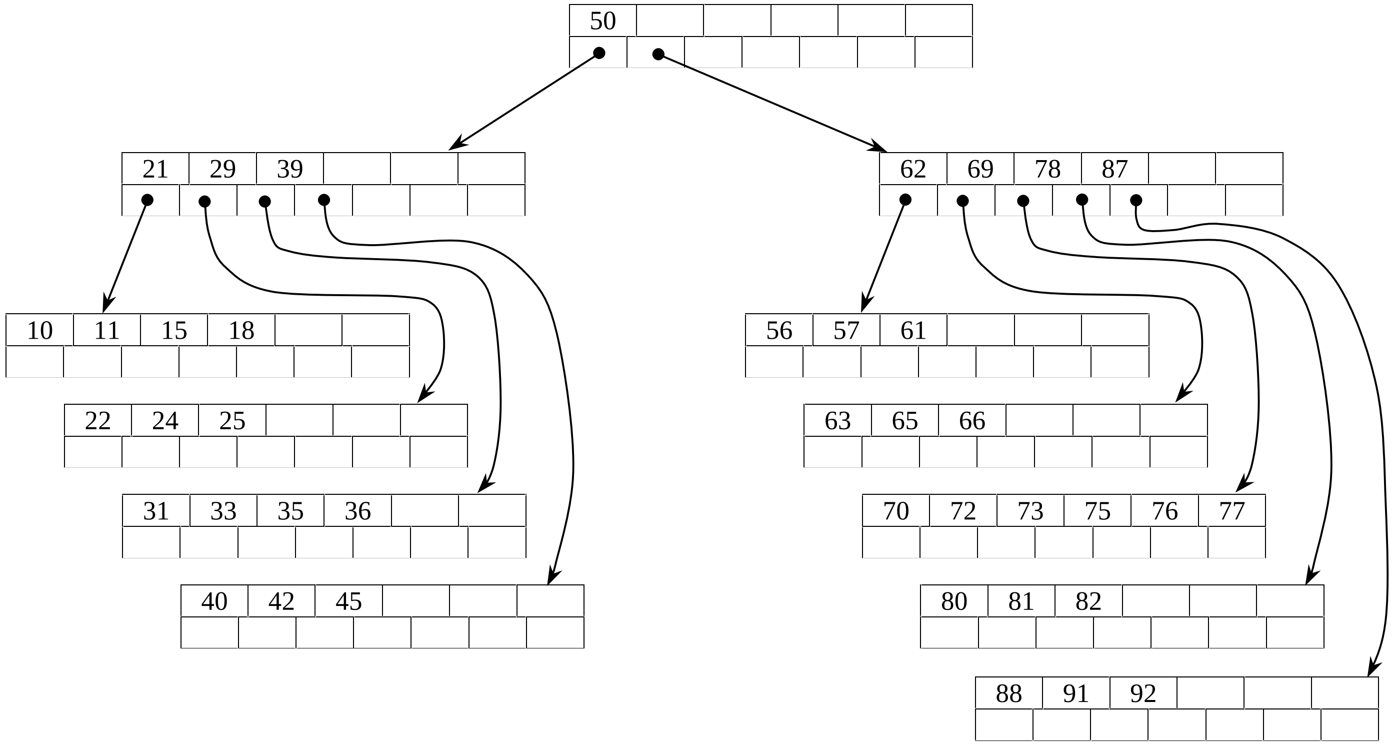
9. ***Add the key with value 77:*** 77 is greater than 50 so it must be inserted in 50’s right subtree. It is greater than 69 but less than 78, so it must be inserted in the root’s right child’s third child (!). There is room in that leaf, so we can just go ahead and insert.



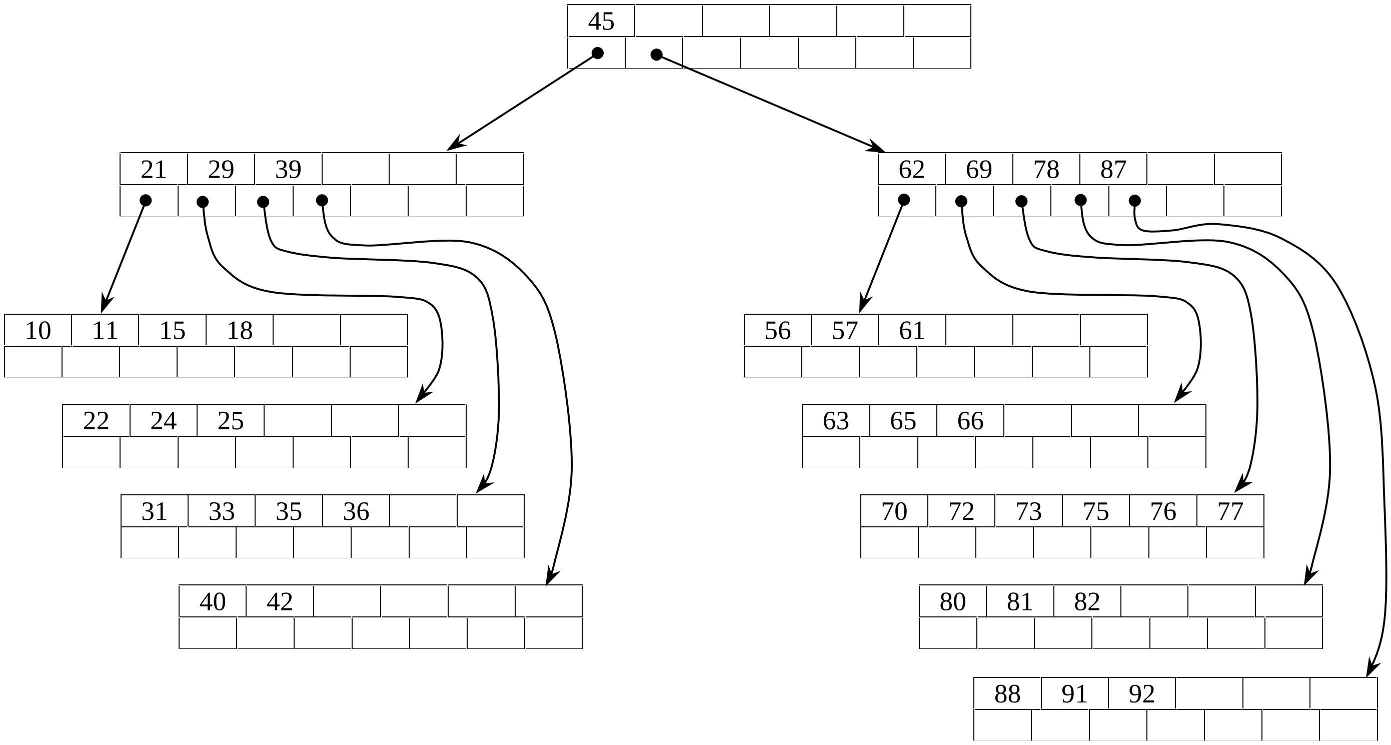
10. ***Add 81:*** 81 is greater than 50 so it belongs in the right subtree. 81 is also greater than 78 so it belongs in the rightmost leaf.



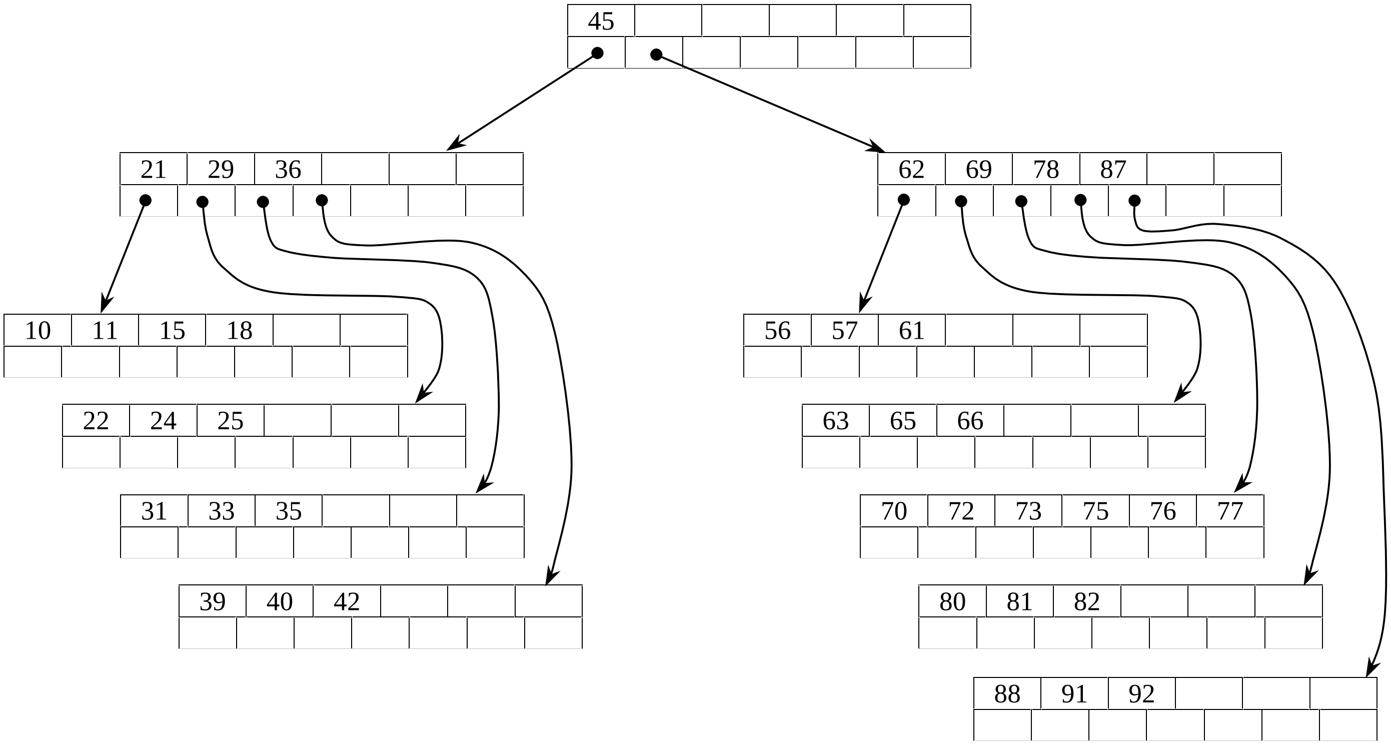
We now have overflow in the rightmost leaf. To repair the overflow, we split the node into left, middle and right parts: 80 81 82 | 87 | 88 91 92. The left part becomes a new node, the right part becomes a new node, and the middle part gets promoted to become a parent.



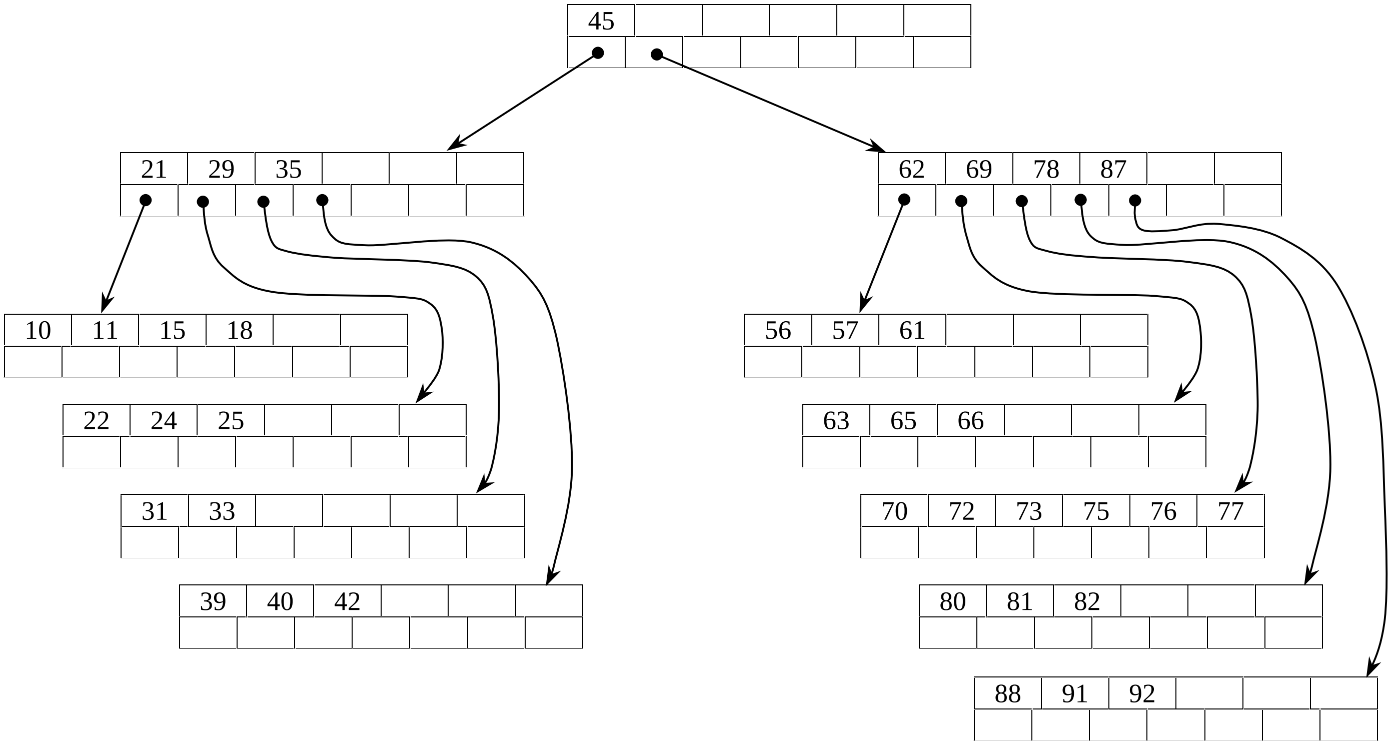
11. ***Delete 50:*** We're only allowed to delete from a leaf, so swap 50 with the largest value in its left subtree (45). Now we can delete 50.



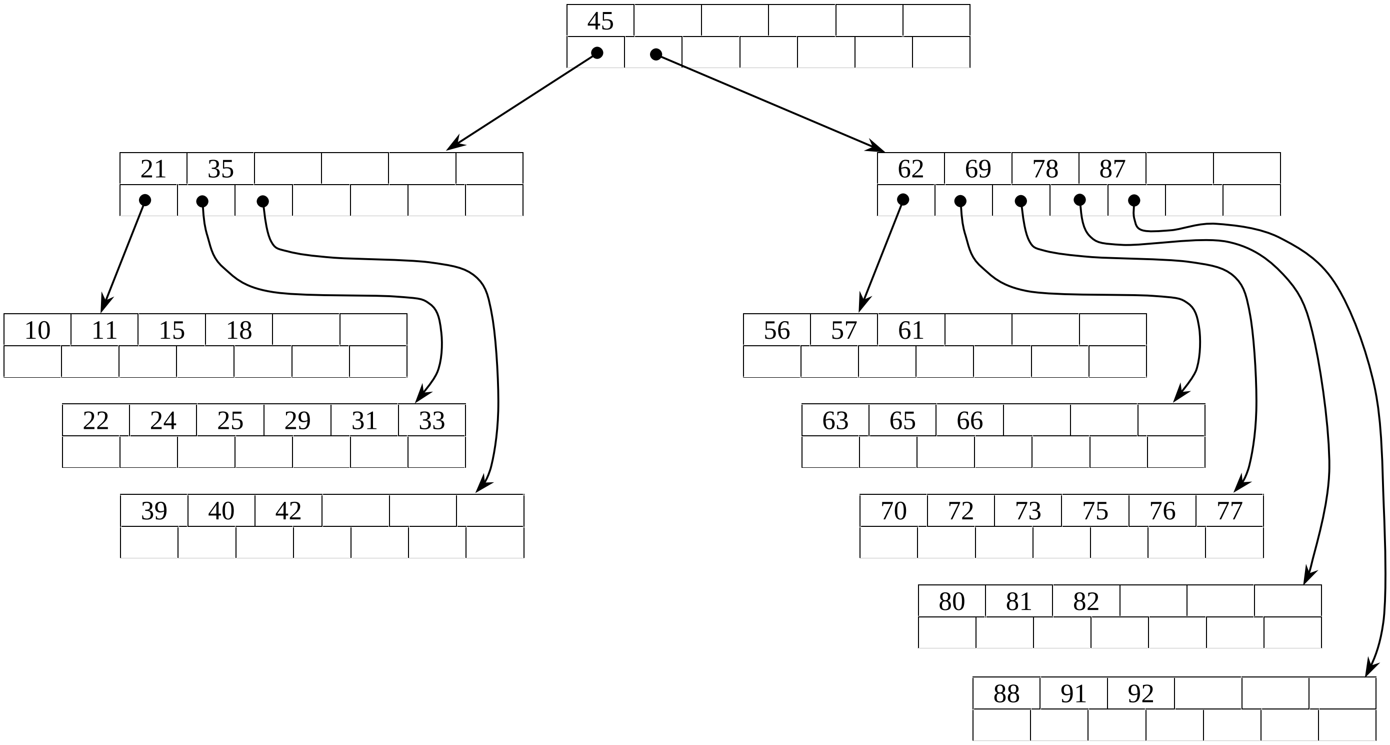
We now have underflow in the rightmost leaf of the left subtree. We combine the node with its more populous neighbour and their parent: 31 33 35 36 39 40 42. We have more than M-1 values, so we must split into left, middle and right: 31 33 35 | 36 | 39 40 42. Left and right become new nodes and middle gets promoted to the parent node.



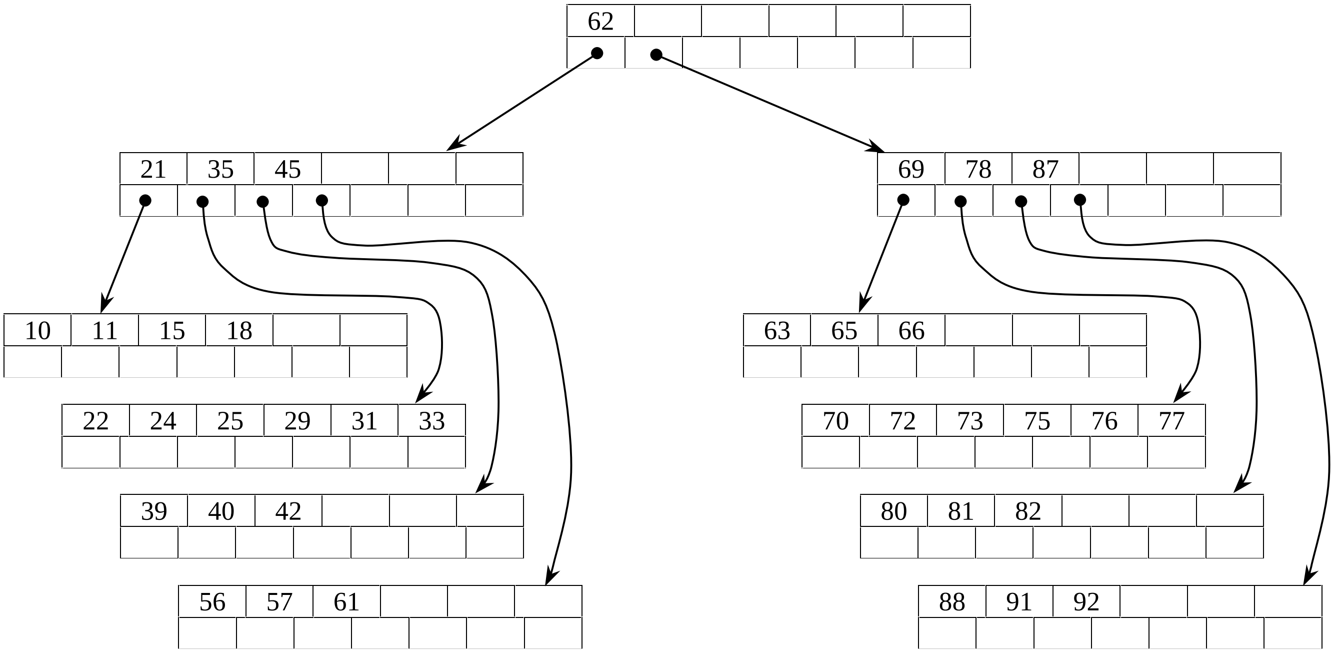
12. Delete 36: Again, 36 is not in a leaf, so we swap it with the largest value in its left subtree, then delete it.



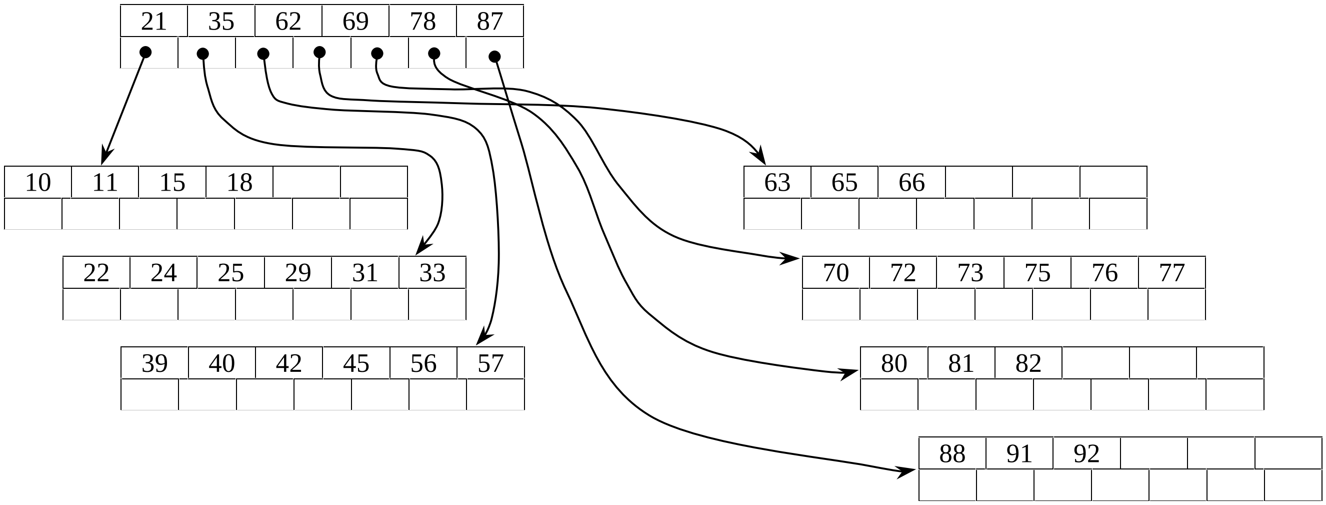
We now have underflow, so we combine the underflowing node with its more populous neighbour (either one in this case) and their parent: 22 24 25 29 31 33. Since the combination contains only M-1 values, we can keep them in a single node.



Repairing the underflow created underflow in the parent. We repair it the same way we repair a leaf node: combine it with its more populous neighbour and their parent: 21 35 45 62 69 78 87. Since there are more than M-1 values, we split into left, middle and right: 21 35 45 | 62 | 69 78 87. Left and right are new nodes, middle is promoted to the parent.



13. ***Finally, delete 61:*** 61 is already in a leaf so we can just delete it. But deleting it causes underflow in the rightmost leaf of the left subtree. We repair the underflow by combining with the neighbour and their parent: 39 40 42 45 56 57. Since there are only M-1 values, this combination becomes a single node, causing underflow in the parent (21 35). To repair the underflow in the parent, combine it with its neighbour and their parent: 21 35 62 69 78 87. Since there are only M-1 values, this combination becomes a single node: the new root of the tree!


21 35 62 69 78 87
10 11 15 18
22 24 25 29 31 33
39 40 42 45 56 57
63 65 66
70 72 73 75 76 77
80 81 82
88 91 92