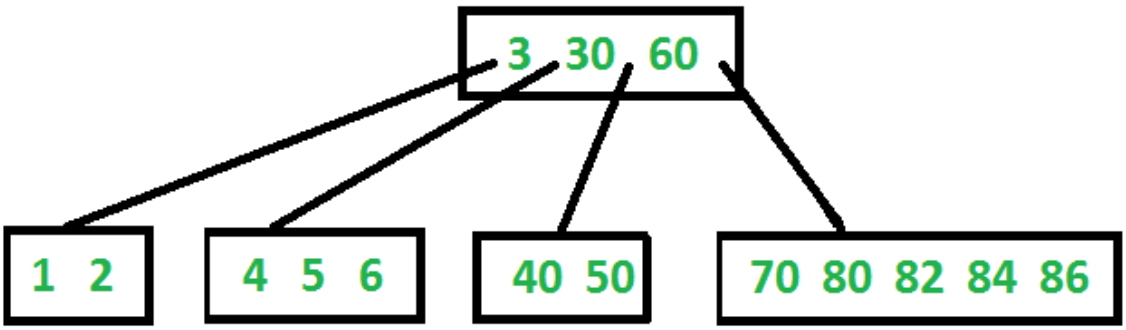
# Assignment 3:

**Installation**

1. To run the file, on a command prompt type ‘python btree.py’

**Explanation**

1. A B-Tree is a balanced m-way tree.
2. A B-Tree of order m satisfies the given properties:
    a. Each node has at most m children
    b. Each internal node has at least m/2 children
    c. Root has at least 2 children if it is not a leaf
    d. A non-root leaf node with k children has at least k-1 keys
    e. All leaves appear in same level
3. Each internal node’s keys act as separation values which divide its subtrees. For example, if an internal node has 3 child nodes (or subtrees) then it must have 2 keys: $a_1$ and $a_2$. All values in the leftmost subtree will be less than $a_1$, all values in the middle subtree will be between $a_1$ and $a_2$, and all values in the rightmost subtree will be greater than $a_2$
4. Following is an example B-Tree of minimum degree 3. Note that in practical B-Trees, the value of minimum degree is much more than 3.




5. **Search in a BTree:** Search is similar to search in Binary Search Tree. Let the key to be searched be k. We start from root and recursively traverse down. For every visited non-leaf node, if the node has key, we simply return the node. Otherwise we recur down to the appropriate child (The child which is just before the first greater key) of the node. If we reach a leaf node and don’t find k in the leaf node, we return NULL.
6. **Traversal in a BTree:** Traversal is also similar to Inorder traversal of Binary Tree. We start from the leftmost child, recursively print the leftmost child, then repeat the same process for remaining children and keys. In the end, recursively print the rightmost child.
7. **Insertions in a BTree:** All insertions start at a leaf node. To insert a new element, search the tree to find the leaf node where the new element should be added. Insert the new element into that node with the following steps:

    If the node contains fewer than the maximum allowed number of elements, then there is room for the new element. Insert the new element in the node, keeping the node's elements ordered.

Otherwise the node is full, evenly split it into two nodes so:

a. A single median is chosen from among the leaf's elements and the new element.
b. Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value.
c. The separation value is inserted in the node's parent, which may cause it to be split, and so on. If the node has no parent (i.e., the node was the root), create a new root above this node (increasing the height of the tree).

8. **Advantages of BTree:**
    a. Keeps keys in sorted order for sequential traversing
    b. Uses a hierarchical index to minimize the number of disk reads
    c. Uses partially full blocks to speed insertions and deletions
    d. Keeps the index balanced with a recursive algorithm
9. Our BTree implementation consists of 3 levels

```
# Construct a BTree of max degree 3
b_tree = BTree(2)
```

We then start inserting the elements in the BTree using the insertion algo defined above and keep printing the order of the tree using the traversal algorithm specified above

```
print('Inserting 2 into the tree, the Tree Structure is now:\n')
b_tree.print_order()
b_tree.insert(2)
```

We then print out the sorted list of all the elements in the BTree

```
# Print Sorted list of elements in the BTree
print('Sorted List of elements in the BTree')
b_tree.print_sorted_elements()
```

We then implement a search that prints out the keys which have been traversed using the search BTree algo specified above

```
# Search Element
print('Search for key, while printing elements that have been
traversed')
b_tree.search(11)
```

The following is the output of our program

```
Inserting 2 into the tree, the Tree Structure is now:
[2]
Inserting 1 into the tree, the Tree Structure is now:
[1, 2]
Inserting 5 into the tree, the Tree Structure is now:
[1, 2, 5]
Inserting 7 into the tree, the Tree Structure is now:
[2]
[1] [5, 7]
Inserting 3 into the tree, the Tree Structure is now:
[2]
[1] [3, 5, 7]
Inserting 11 into the tree, the Tree Structure is now:
[2, 5]
[1] [3] [7, 11]
Sorted List of elements in the BTree
1 2 3 7 11

The BTree
[2, 5]
[1] [3] [7, 11]

Search for key, while printing elements that have been traversed
Checking if value is equal to 2
Checking if value is equal to 5
Checking if value is equal to 7
Checking if value is equal to 11

Process finished with exit code 0
```

The shape of our BTree is as follows: