CS 225

Welcome to Lab BTrees!

lab_btree **Belligerent BTrees**

1. B-Tree Definition
2. Insertion
3. Implementation

B-Tree Definition

A B-Tree of order m

1. Maintains ordering within nodes
2. Each node has one more child than keys
3. All leaves are on the same level

Order = m	Possible number of keys	Possible number of children
Root node	[1, m-1]	[2, m]
Non-root nodes	[ceil(m/2)-1, m-1]	[ceil(m/2), m]

Order = 5	Possible number of keys	Possible number of children
Root node		
Non-root nodes		

B-Tree Definition

A B-Tree of order m

1. Maintains ordering within nodes
2. Each node has one more child than keys
3. All leaves are on the same level

Order = m	Possible number of keys	Possible number of children
Root node	[1, m-1]	[2, m]
Non-root nodes	[ceil(m/2)-1, m-1]	[ceil(m/2), m]

Order = 5	Possible number of keys	Possible number of children
Root node	[1, 4]	[2, 5]
Non-root nodes		

B-Tree Definition

A B-Tree of order m

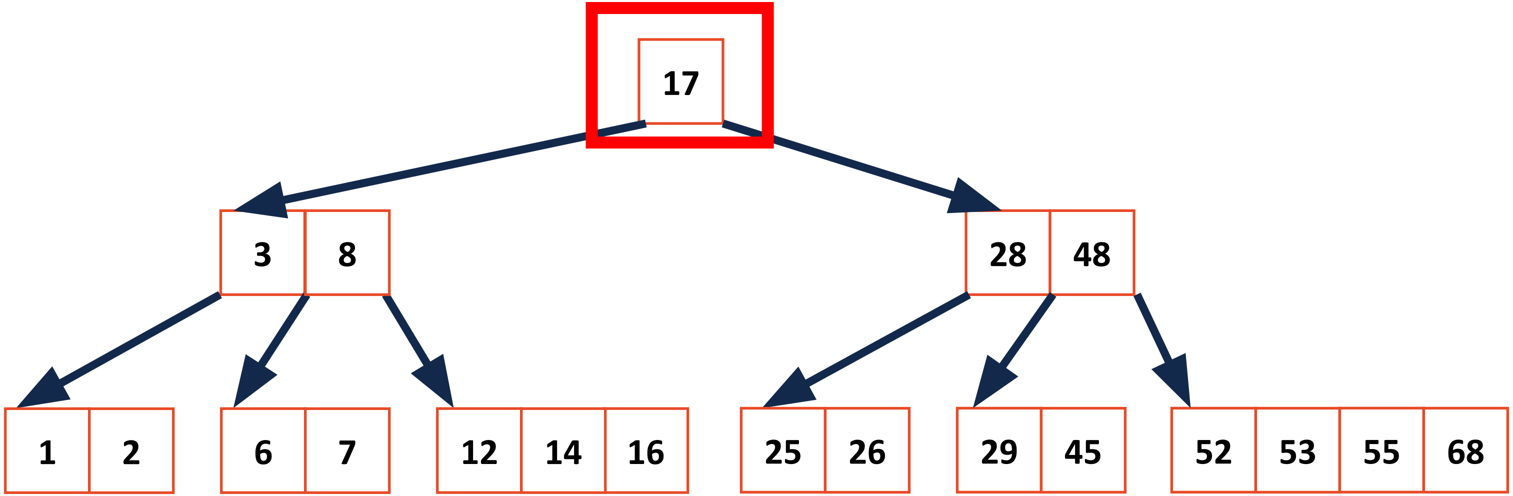
1. Maintains ordering within nodes
2. Each node has one more child than keys
3. All leaves are on the same level

Order = m	Possible number of keys	Possible number of children
Root node	[1, m-1]	[2, m]
Non-root nodes	[ceil(m/2)-1, m-1]	[ceil(m/2), m]

Order = 5	Possible number of keys	Possible number of children
Root node	[1, 4]	[2, 5]
Non-root nodes	[2, 4]	[3, 5]

Btree Properties

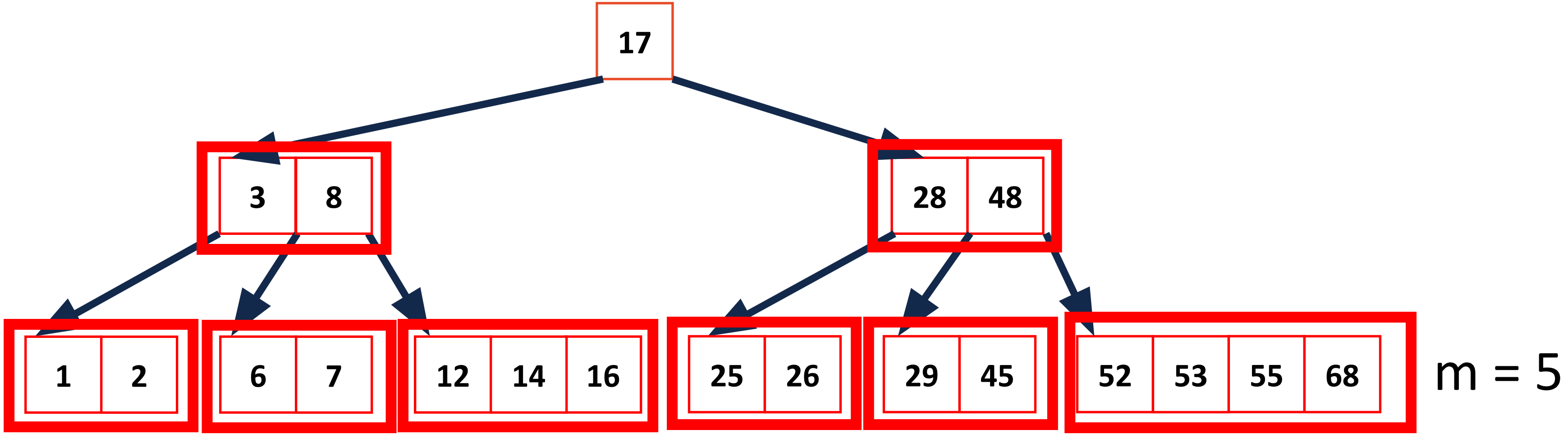
- Root nodes can have: [1, m-1] keys and [2, m] children
- All non-root, nodes have [⌈m/2⌉-1, m-1] keys.
- All non-root internal nodes have [⌈m/2⌉, m] children



m = 5

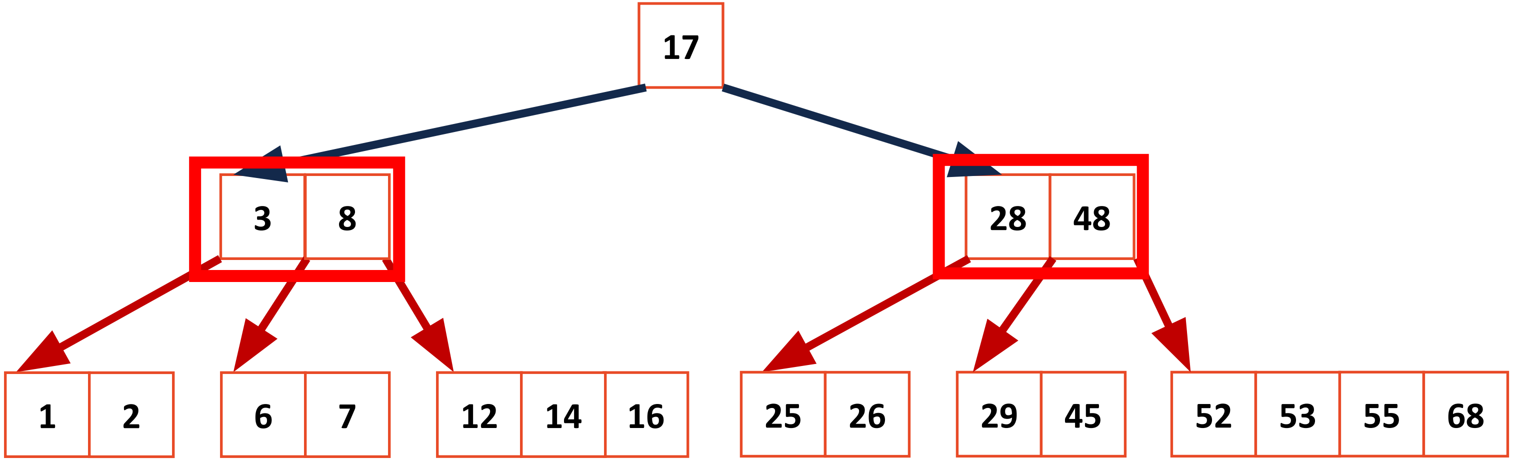
Btree Properties

- Root nodes can have: [1, m-1] keys and [2, m] children
- All non-root, nodes have [⌈m/2⌉-1, m-1] keys.
- All non-root internal nodes have [⌈m/2⌉, m] children



Btree Properties

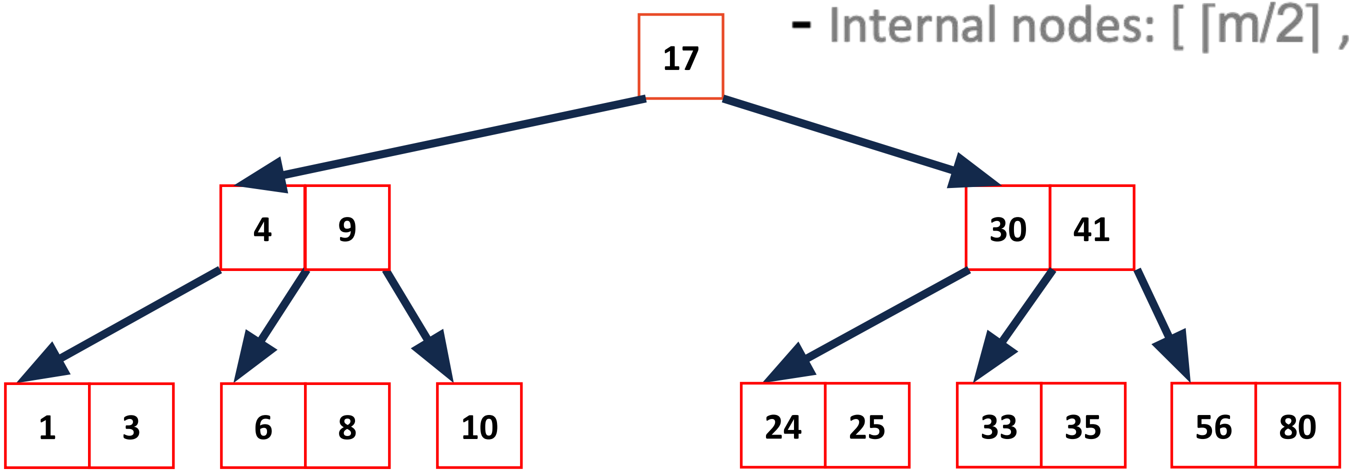
- Root nodes can have: [1, m-1] keys and [2, m] children
- All non-root, nodes have [$\lceil m/2 \rceil$-1, m-1] keys.
- All non-root internal nodes have [$\lceil m/2 \rceil$, m] children



m = 5

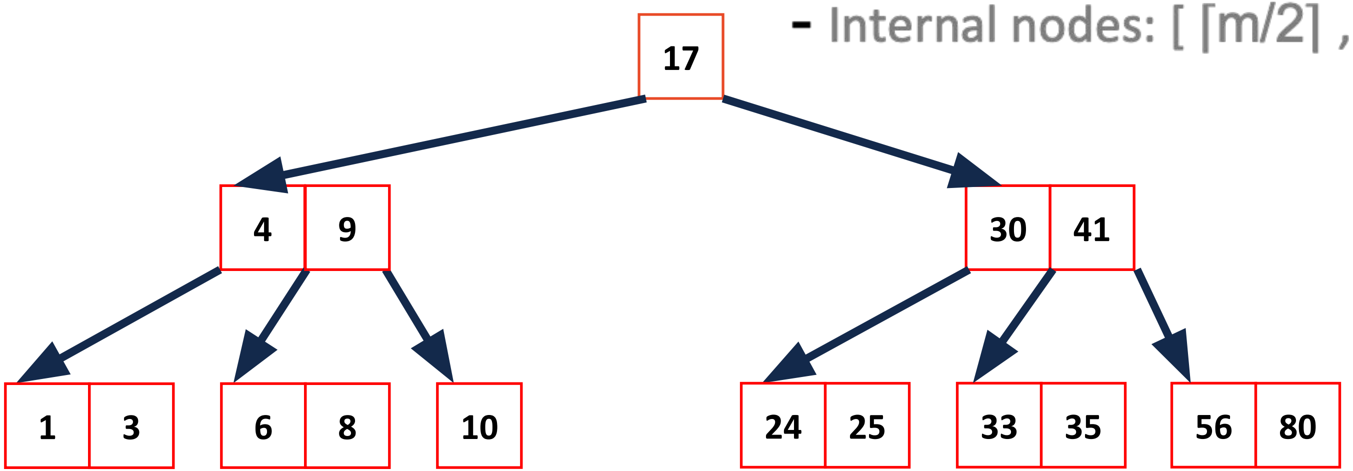
What are the possible values of m if the following B-Tree is of order m?

- [[m/2] -1, m-1] keys
- Internal nodes: [[m/2] , m] children



What are the possible values of m if the following B-Tree is of order m?

- [[m/2] -1, m-1] keys
- Internal nodes: [[m/2] , m] children



1. $m \geq 3$ // we have nodes with three children
2. We have node with just one element, that means

$$ceil\left(\frac{m}{2}\right) - 1 = 1 \Rightarrow ceil\left(\frac{m}{2}\right) = 2 \Rightarrow m = 3, 4$$

1 is the minimum number of elements node can have, hence (=)

Which B-tree is a valid B-tree?

```
   _C I_
  /  |  \
AB   F   JK
    / \
   DE GH
```

1

```
   _C E_
  /  |  \
AB   D   FGHI
```

2

```
       _D I_
      /  |  \
    B    F    J M
   / \  / \    | \
  A  C E  GH  KL  N
```

3

```
   _C F_
  /  |  \
AB   DE  GHI
```

4

Which B-tree is a valid B-tree

```
     _C I_
    /  |  \
 AB    F   JK
      / \
    DE  GH
```

1

```
     _C E_
    /  |  \
 AB    D   FGHI
```

2

```
          _D I_
         /  |  \
       B    F    J M
      / \  / \    | \
     A  C E  GH  KL  N
```

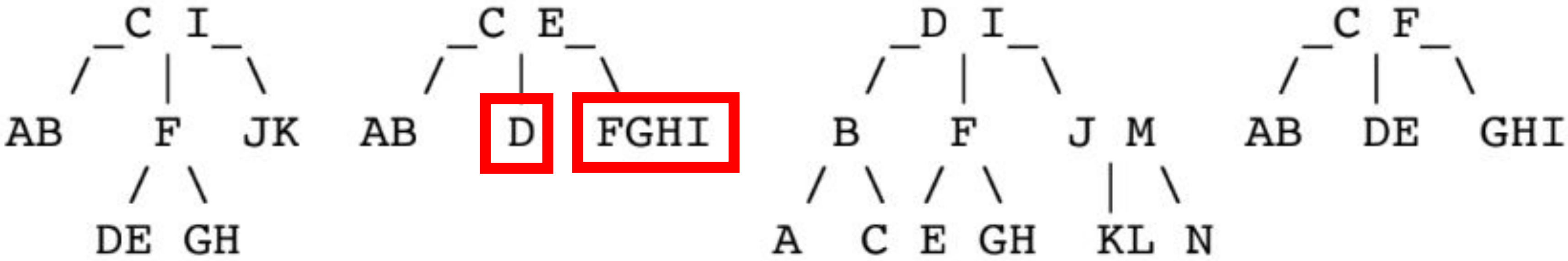
3

```
     _C F_
    /  |  \
 AB   DE   GHI
```

4

1. Leaves are on different level

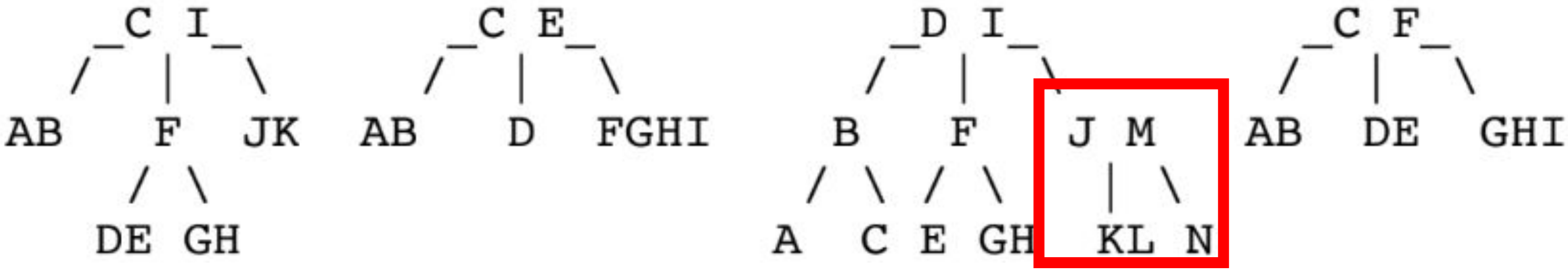
Which B-tree is a valid B-tree



~~1~~ 2 3 4

2. m should be at least 5, but we have node with only one element in it.

Which B-tree is a valid B-tree



~~1~~ ~~2~~ 3 4

3. Node ‘JM’ does not have enough children

Which B-tree is a valid B-tree

```
     _A  G_          _A  G_           _A  G_          _A  G_
    /   |   \       /   |   \        /   |   \       /   |   \
BC      D   EF  BC      D   EFHI    C     D     E  F  BC   JD   EFH
       / \                        / \   / \     |  \
      GH  JK                     I   L H  JK   MO  P
```

~~1~~ ~~2~~ ~~3~~ 4

4. B-tree: m = 4, 5, 6 √

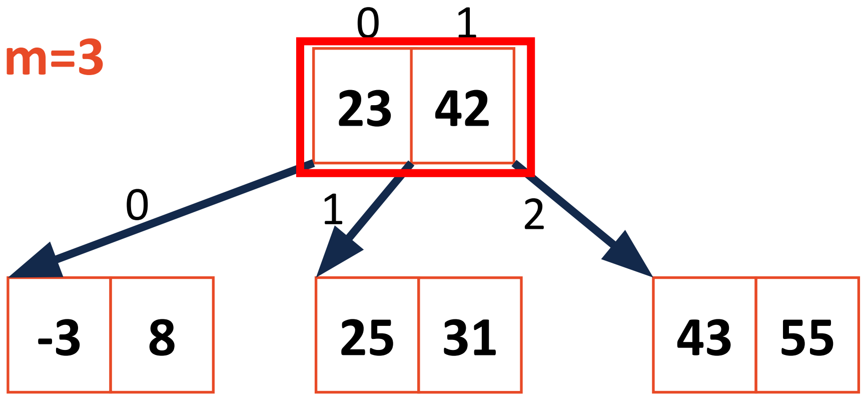
- [ceil(m/2)-1, m-1] keys
- Internal nodes: [ceil(m/2), m] children

Insertion

1. Insert the new element into a leaf node (keeping sorted property).

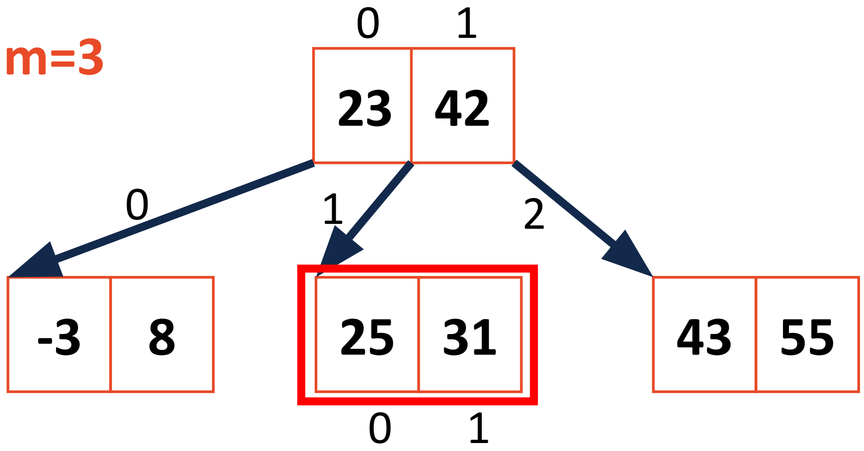
2. If the leaf node has more than m-1 elements: we must ***split*** the node by ***throwing up*** the middle element to the parent node. Check parent node for overflow;

Insert(28)



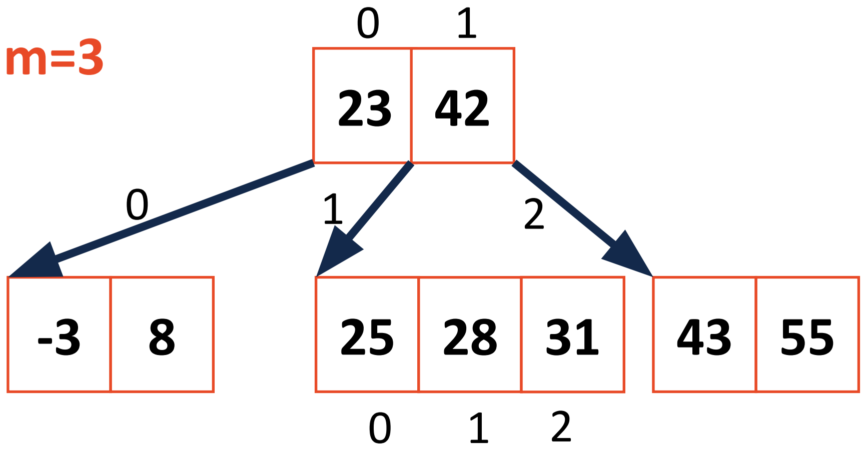
1. Find the corresponding leaf node, start from the root:
   - 23 < **28** <42 ⟹ follow pointer 1;

Insert(28)



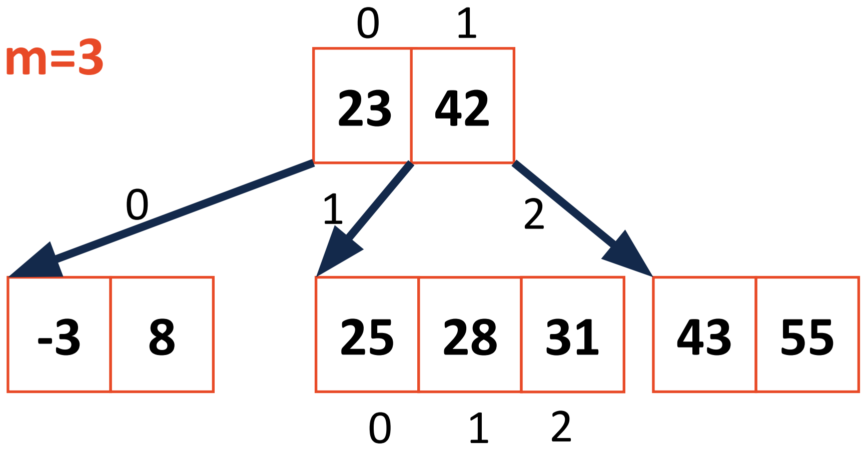
3. Current node is the leaf - How do you know?
4. Insert 28 on the right index: 25 < **28** <31

Insert(28)



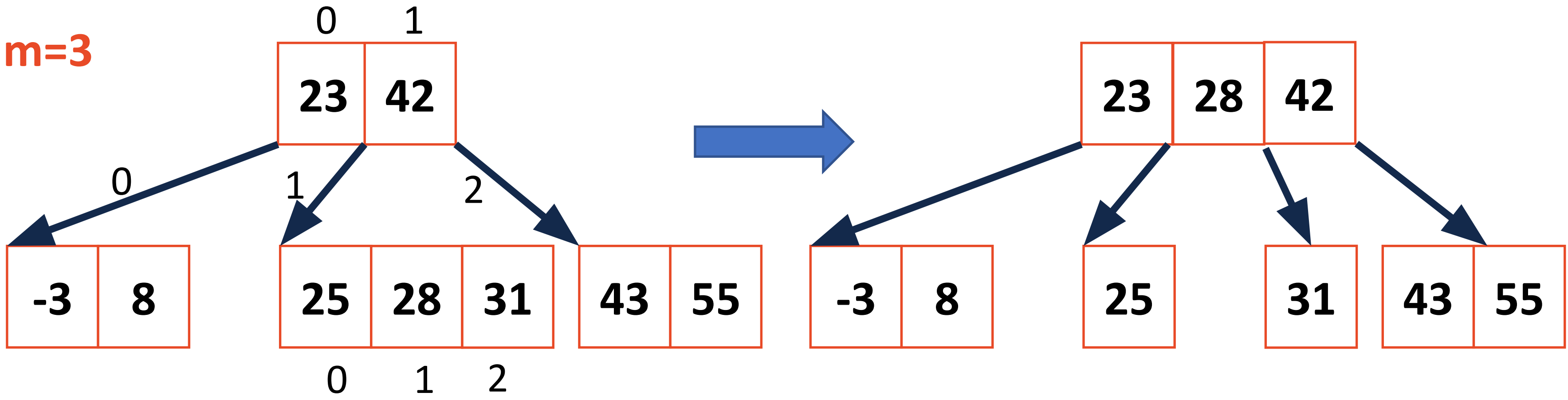
3. Current node is the leaf
4. Insert 28 on the right index: 25 < **28** <31
5. Check node for overflow: since m=3, we have to split the node

Insert(28)



3. Current node is the leaf
4. Insert 28 on the right index: 25 < **28** <31
5. Check node for overflow: since m=3, we have to ‘throw up’ middle element and split the node.

Insert(28)

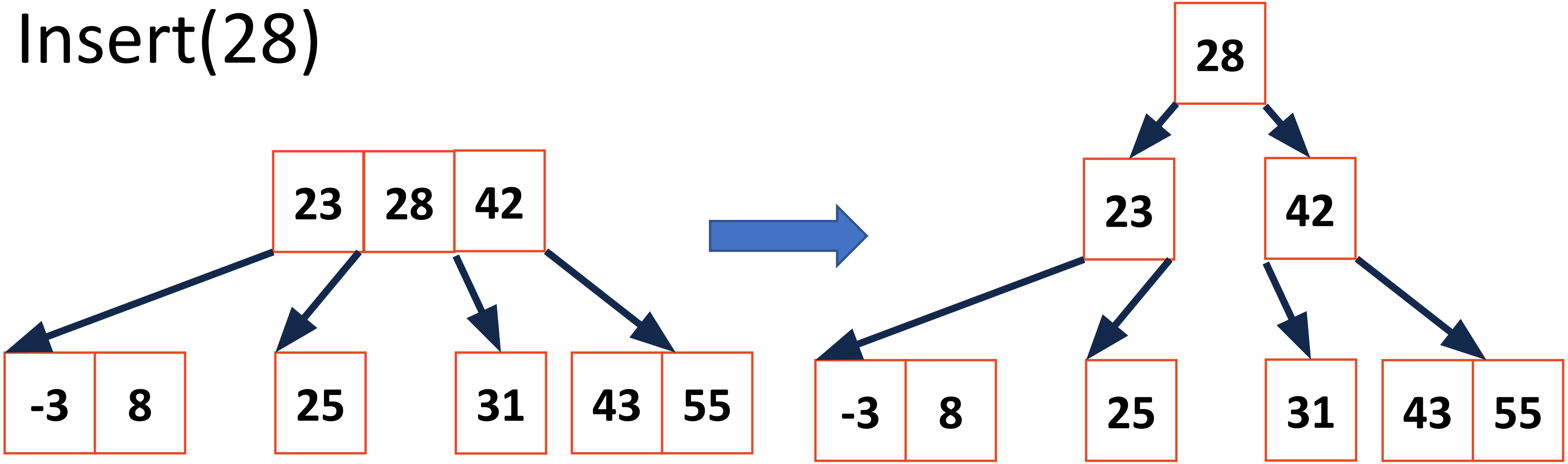


Split node:

Split the node and insert middle element into the parent node;

Check the parent node for overflow => split parent;

Insert(28)

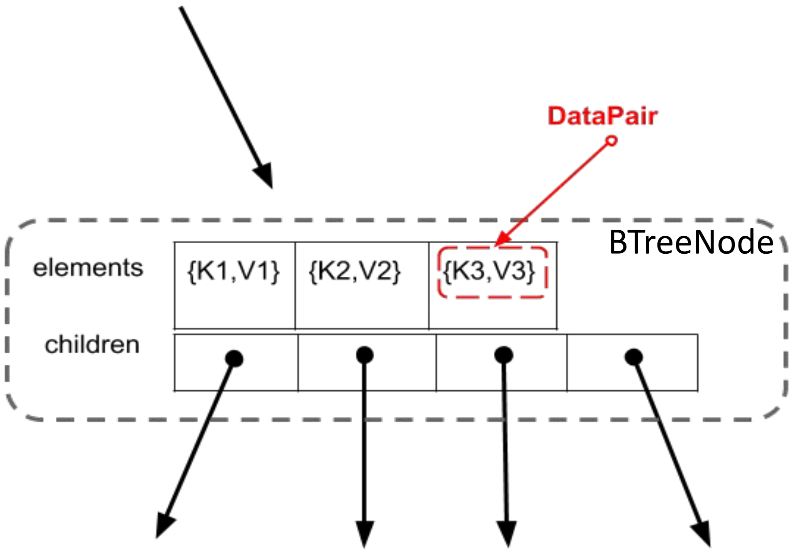


Split node (parent):
Split the node and ‘throw up’ middle element (create new node with 28);
Check new node for overflow => we are done!

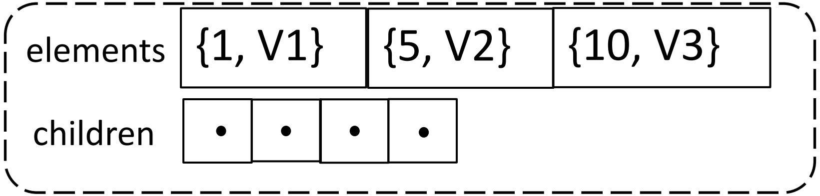
Implementation

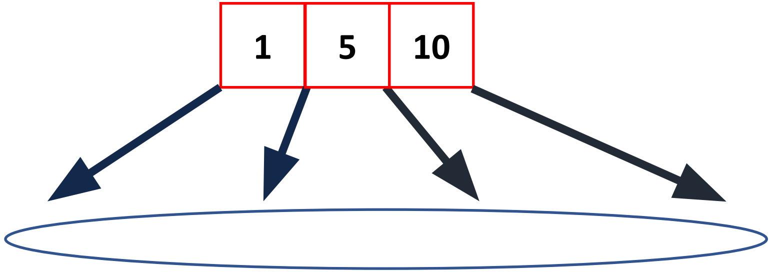
```
class BTree
{
  private:
     struct DataPair {
          K key;
          V value;
    . . .
    };
    struct BTreeNode {
          bool is_leaf;
          vector<DataPair> elements;
          vector<BTreeNode*> children;

          BTreeNode(bool is_leaf, unsigned int order);
          . . .
    };
    unsigned int order;
    BTreeNode* root;  . . . }
```

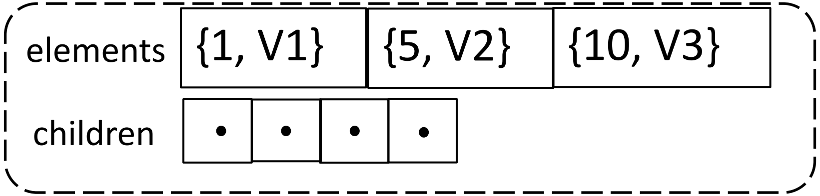


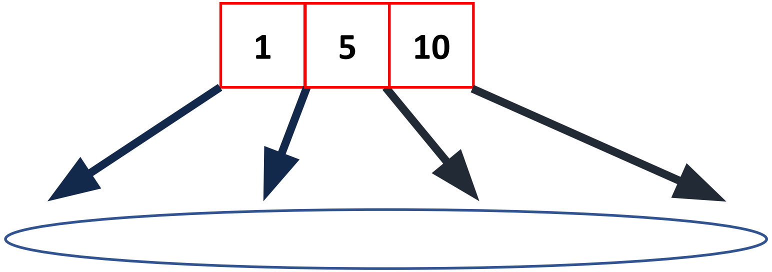
Suppose we are searching for the element with key = 7, which **child pointer** in the BTreeNode should we follow? Give its index in the **children** vector.





Suppose we are searching for the element with key = 7, which **child pointer** in the BTreeNode should we follow? Give its index in the **children** vector.



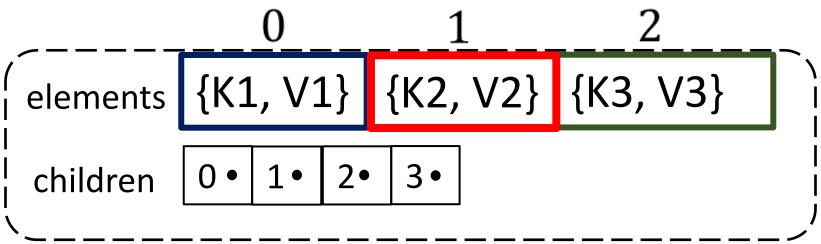


Since 5 < 7 < 10

We need a child node between 5 and 10:

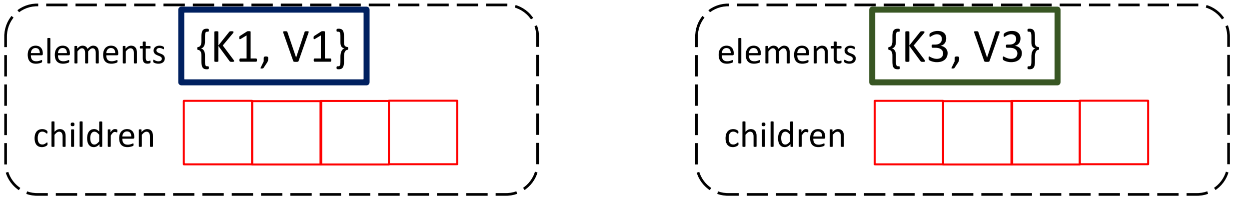
(1 is index of 5 and since 5 < 7 we need child on the right side of the 5, thus index 2)

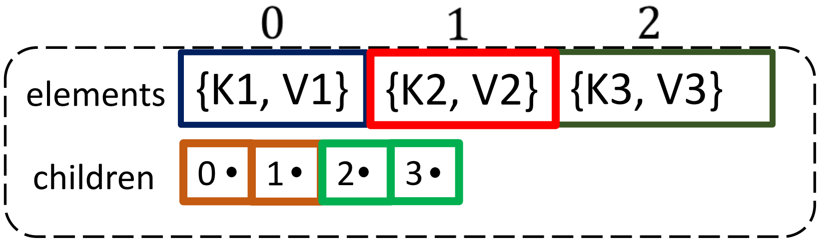
Suppose we need to split the given BTreeNode (so we will throw up the element {K2, V2}) how would we split the children vector between the newly created BTreeNodes after the split?

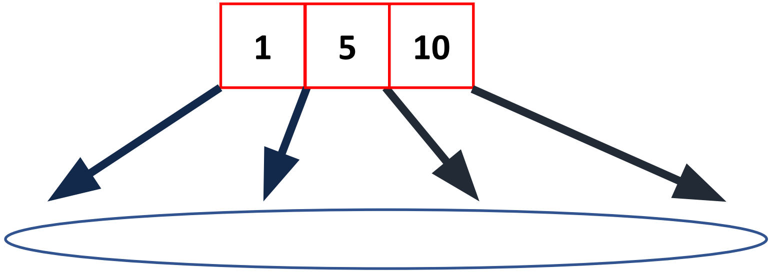


Index of $\{k2, V2\}$ element is $i \ = 1$

<u>Split elements</u>: [0, i), [i +1, end) -> [0,1), [2,3)

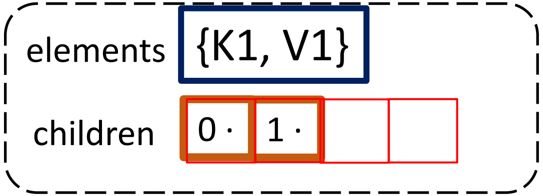


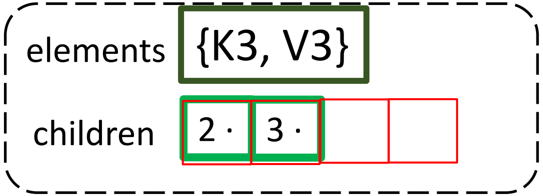




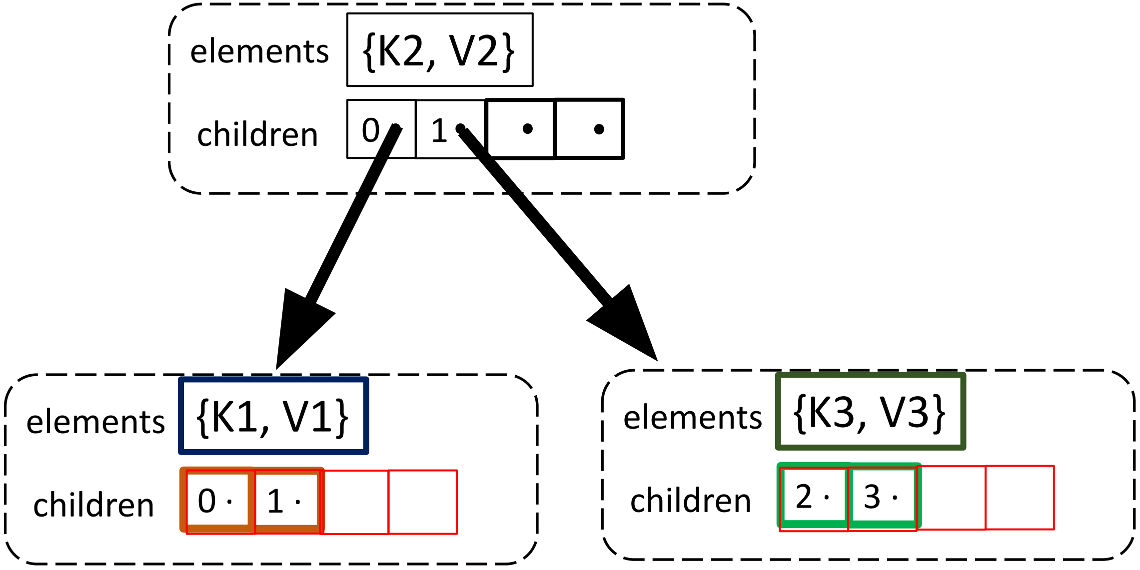
Index of $\{k2, V2\}$ element is $i = 1$

<u>Split children</u>: [0, i], [i +1, end] -> [0,1], [2,3]





Create parent node with {K2, V2}, insert pointers to children nodes



Insertion:

1. Insert the new element into a leaf node (keeping sorted property).

2. Check if you must ***split*** the node by ***throwing up*** the middle element to the parent node. Check parent node for overflow;

Summary

Definition

A B-Tree of order m

1. Maintains ordering within nodes
2. Each node has one more child than keys
3. All leaves are on the same level

```
class BTree
{
  private:
     struct DataPair {
          K key;
          V value;
     . . .
    };
    struct BTreeNode {
          bool is_leaf;
          vector<DataPair> elements;
          vector<BTreeNode*> children;

          BTreeNode(bool is_leaf, unsigned int order);
          . . .
    };
    unsigned int order;
    BTreeNode* root;  . . . }
```

Implementation