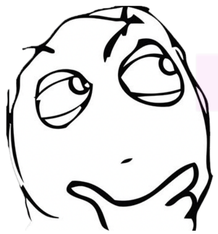
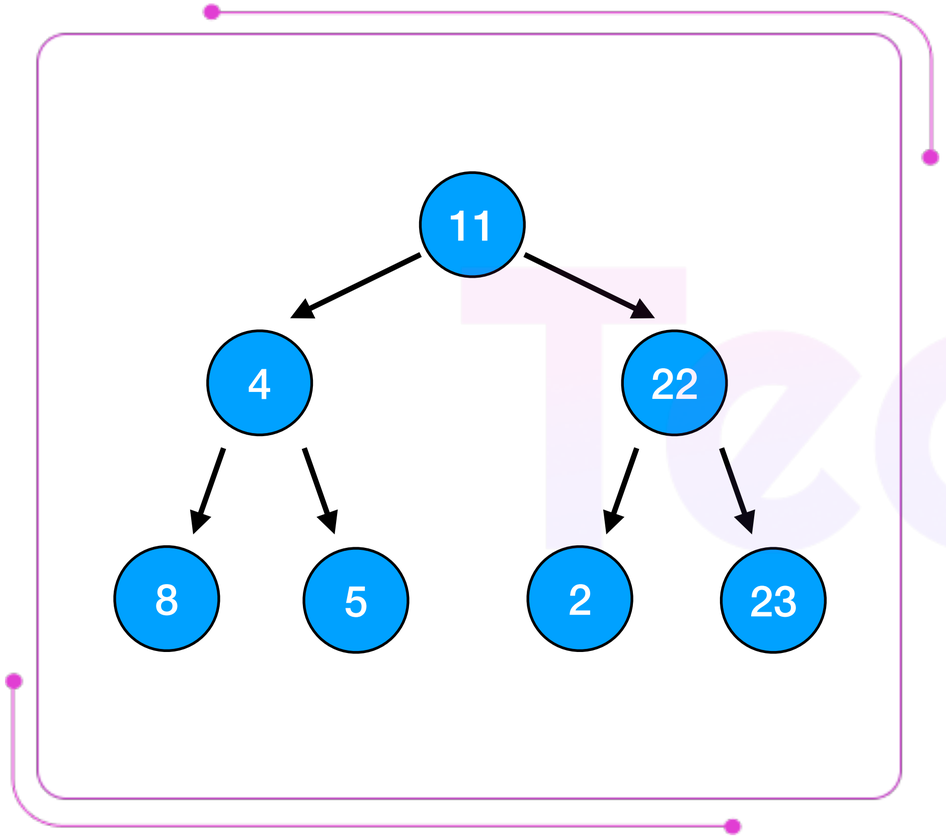


AVL Tree

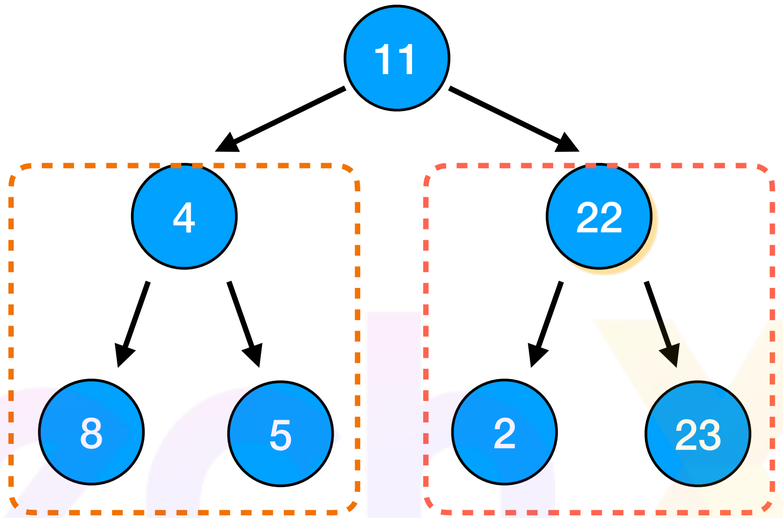


What is AVL Tree ?

- Cây AVL là một dạng cây tìm kiếm nhị phân cân bằng, nơi mỗi nút có một yếu tố cân bằng (chênh lệch chiều cao giữa hai cây con trái và phải) là -1, 0 hoặc 1

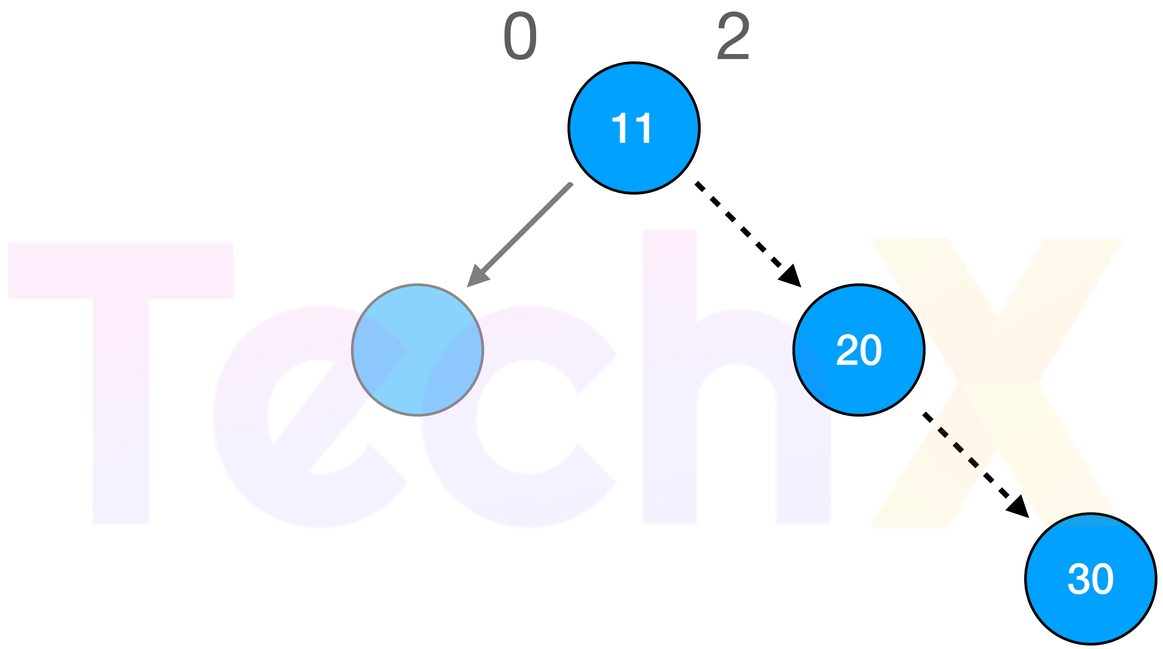
- Khi thêm hoặc xoá một phần tử, nếu cây mất cân bằng, các phép xoay (rotate) sẽ được áp dụng để phục hồi lại tính cân bằng của cây.

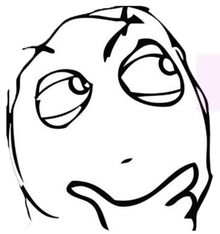
Height of Left Subtree



Height of Right Subtree

$$| \quad - \quad | < 1$$

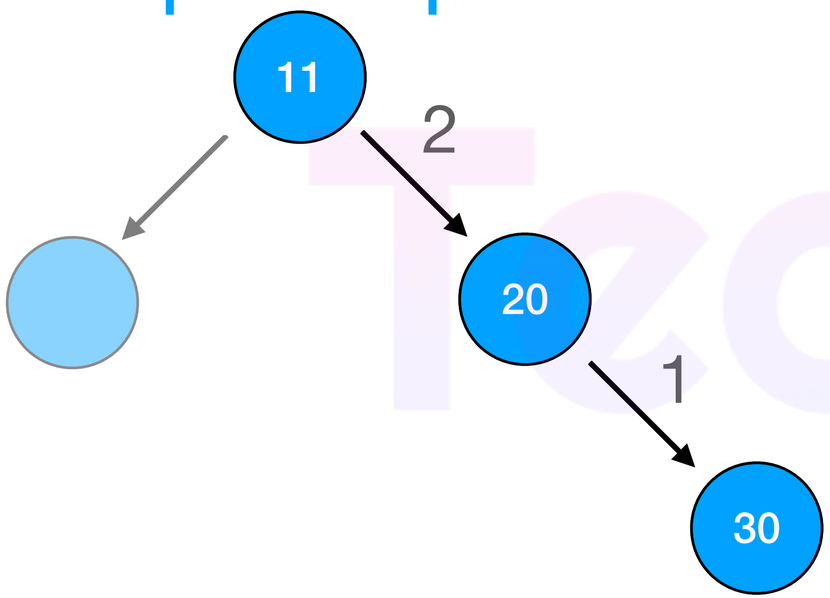

0
2
11
20
30



How many ways are there to rotate ?

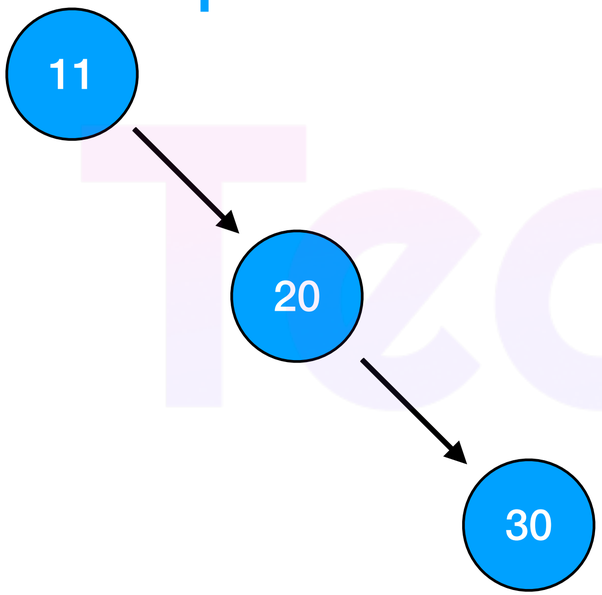
Left Rotation

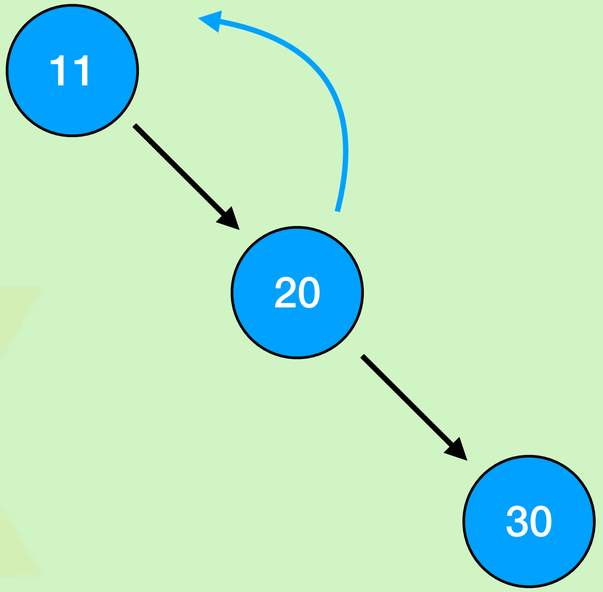
$|0 - 2| > 1$



Left Rotation

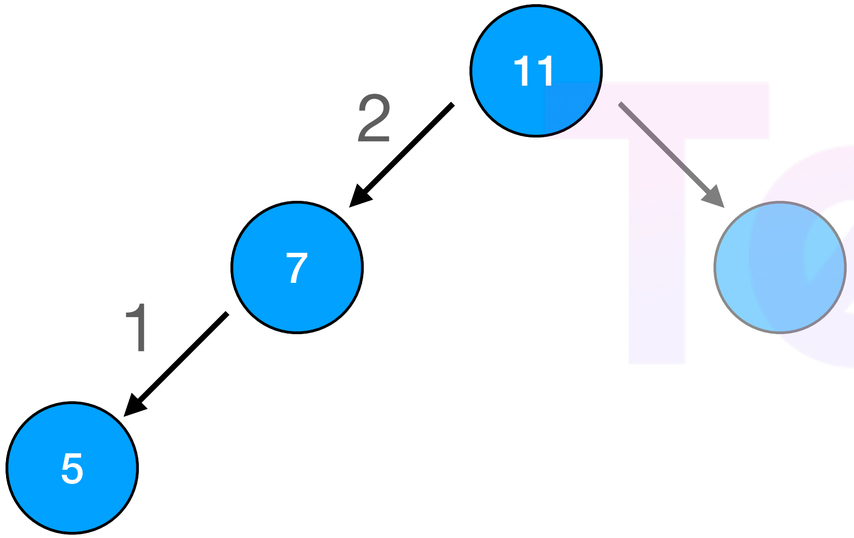
$$|0 - 2| > 1$$





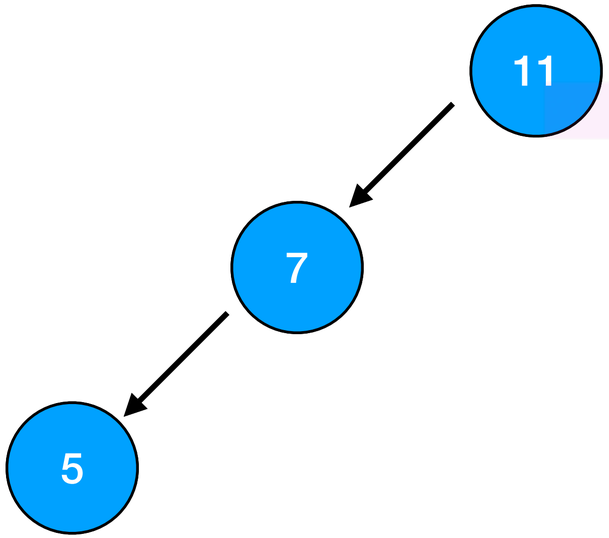
Right Rotation

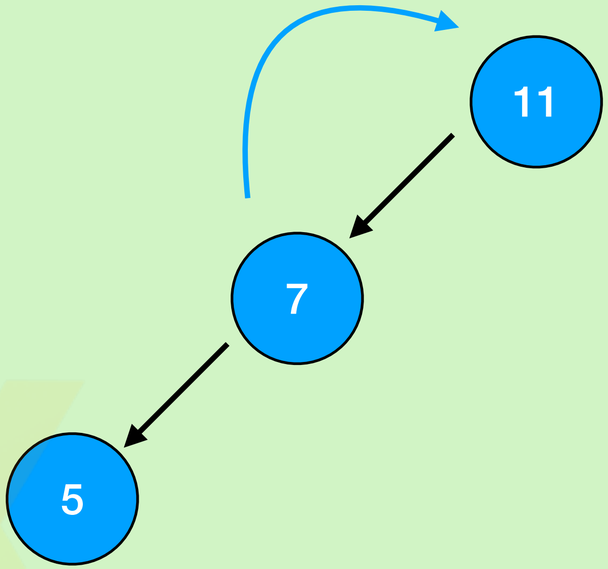
$$|2 - 0| > 1$$



Left Rotation

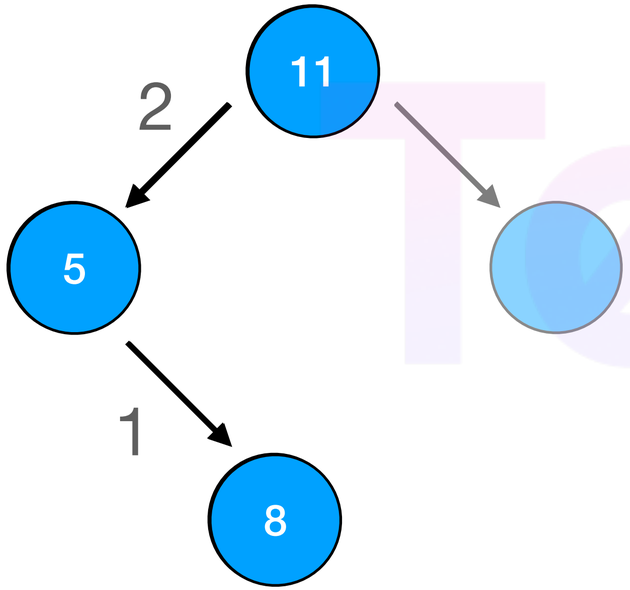
$$|2 - 0| > 1$$





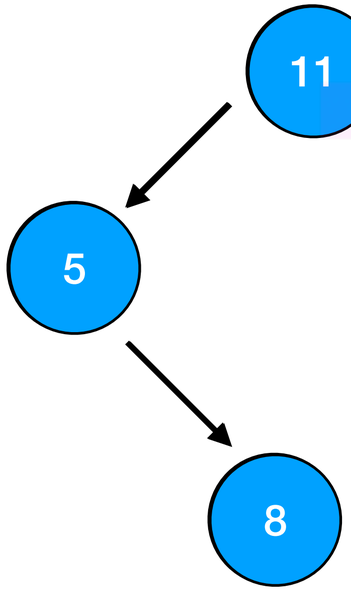
Left right Rotation

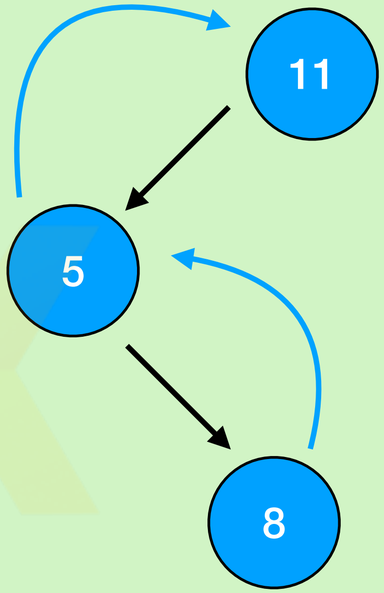
$$|2 - 0| > 1$$



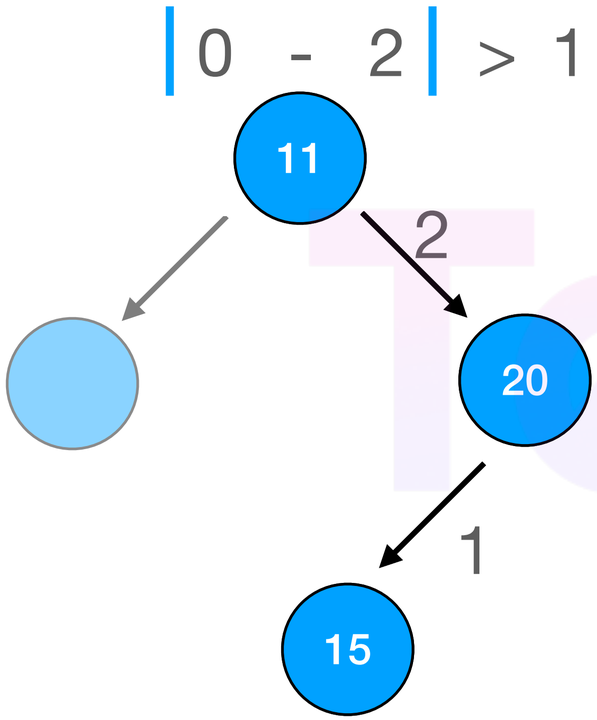
Left right Rotation

$$|2 - 0| > 1$$



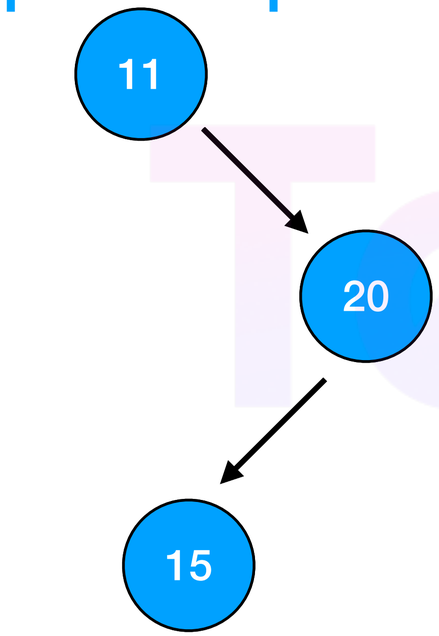


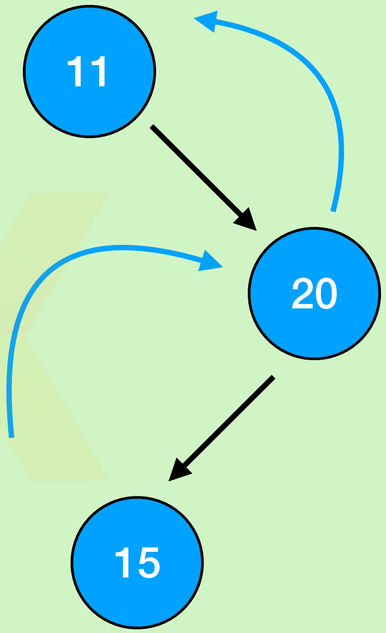
Right Left Rotation



Left Rotation

$|0 - 2| > 1$



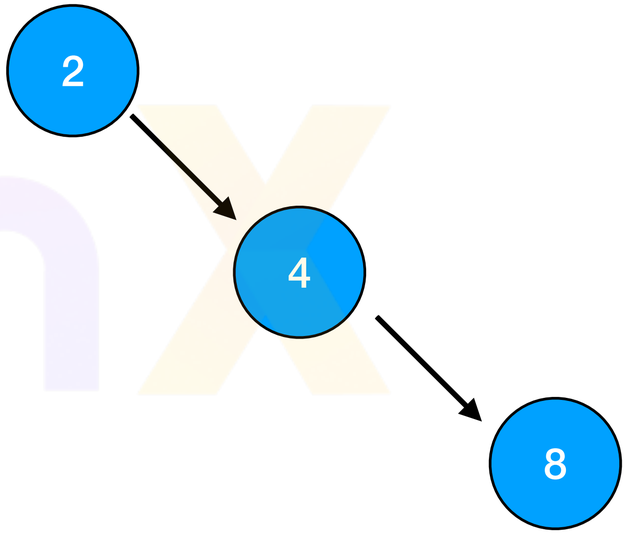


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]

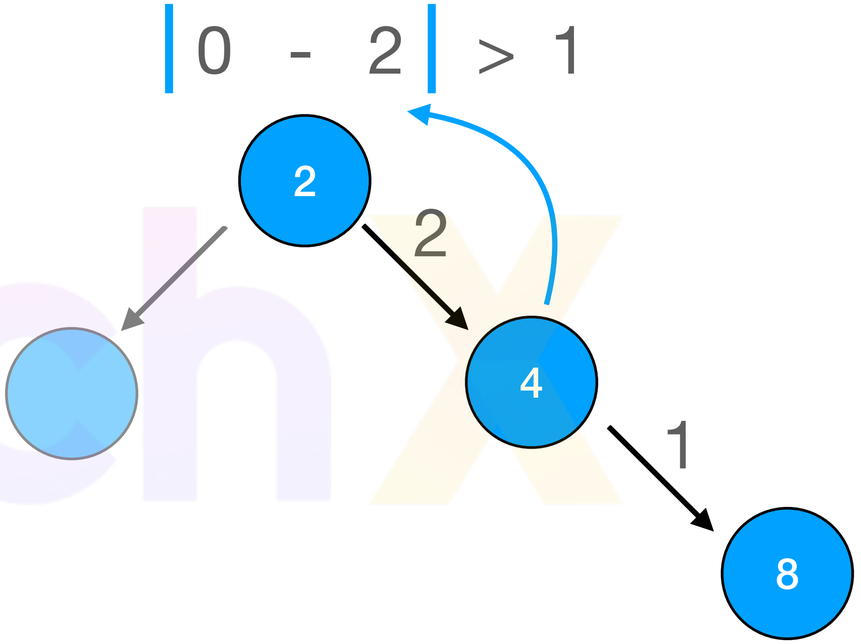


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]

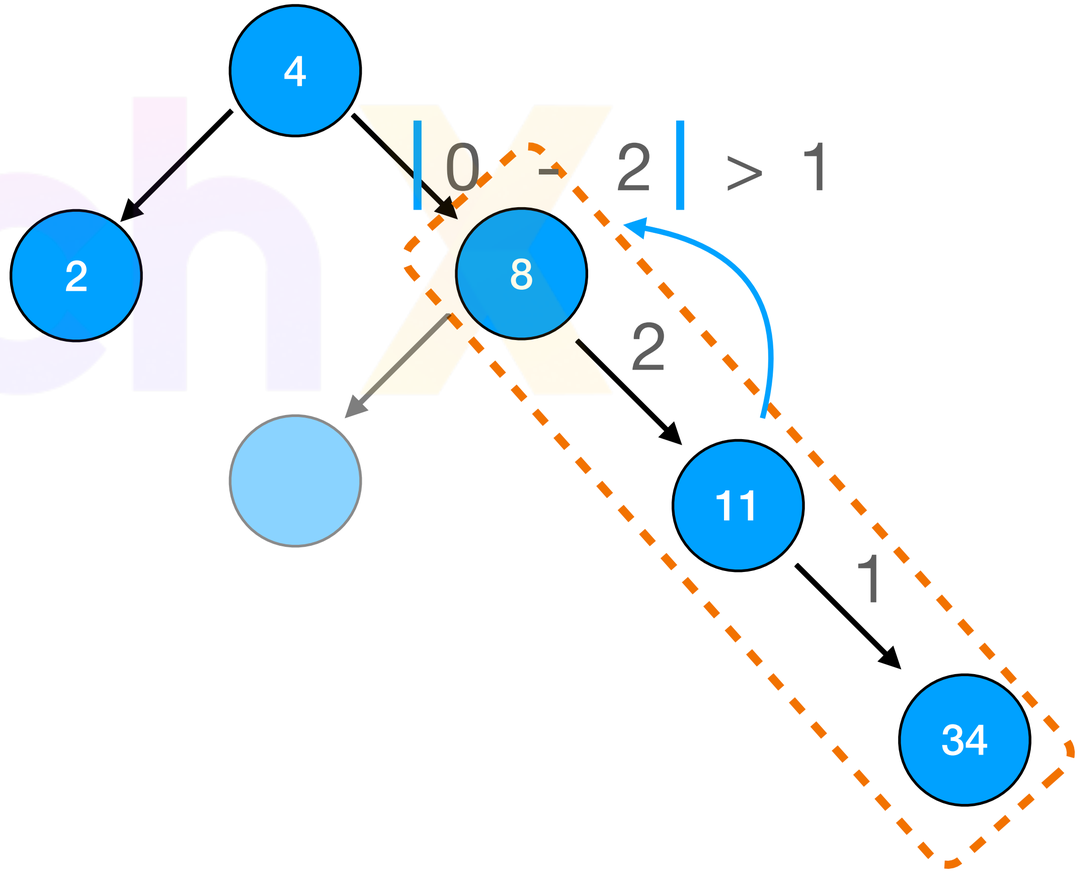


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]

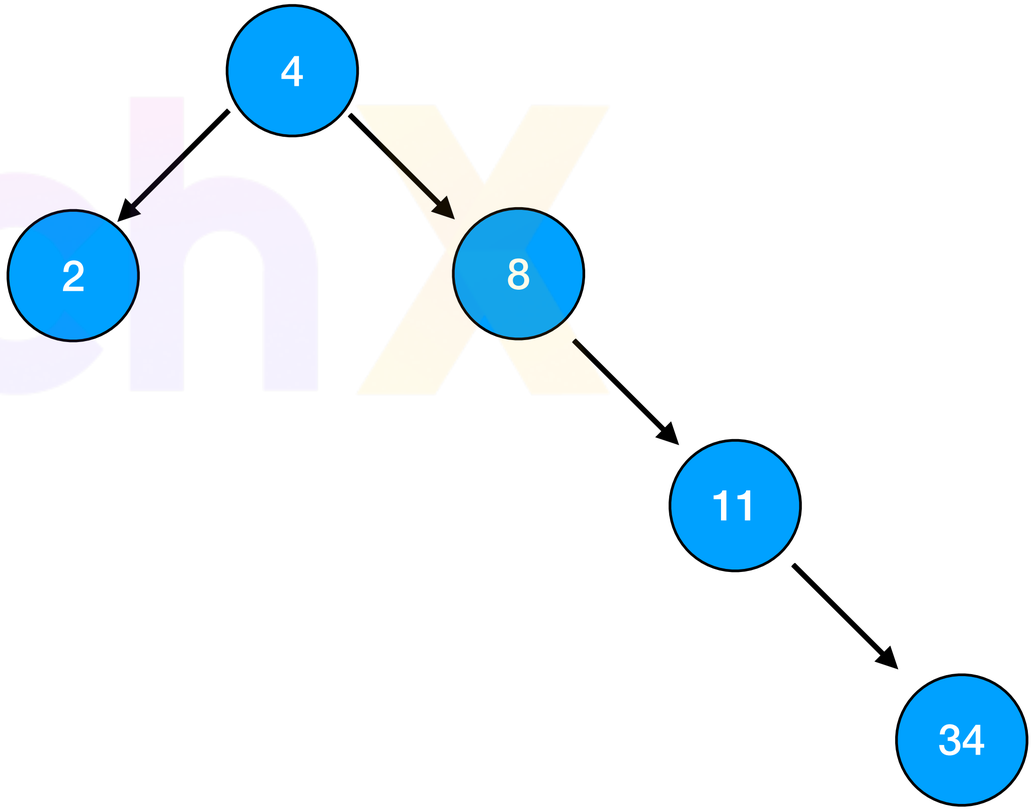


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]

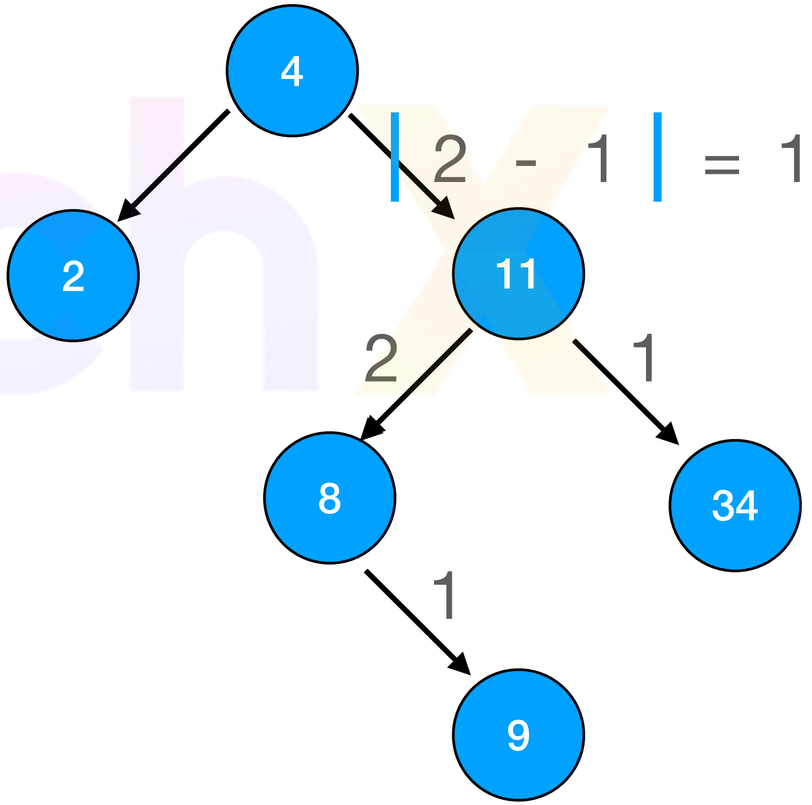


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]



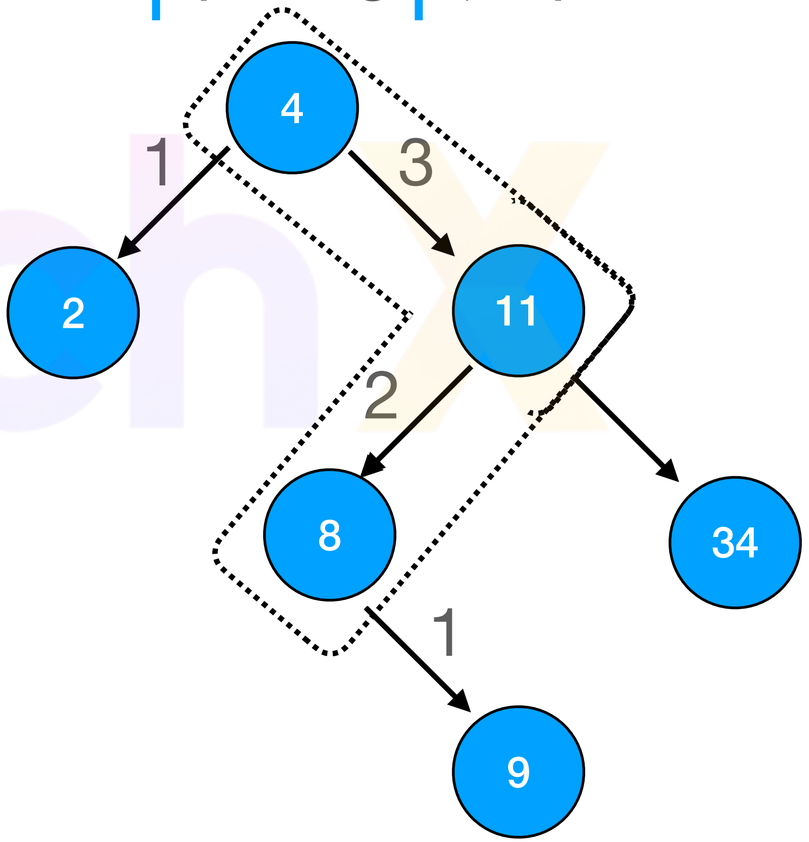
name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]

$|1 - 3| > 1$

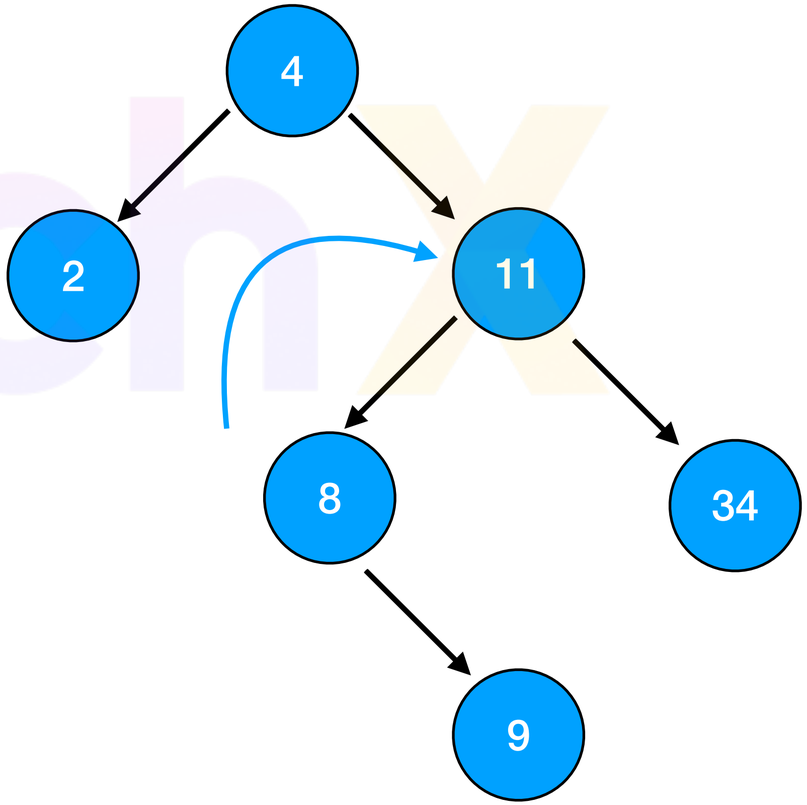


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]

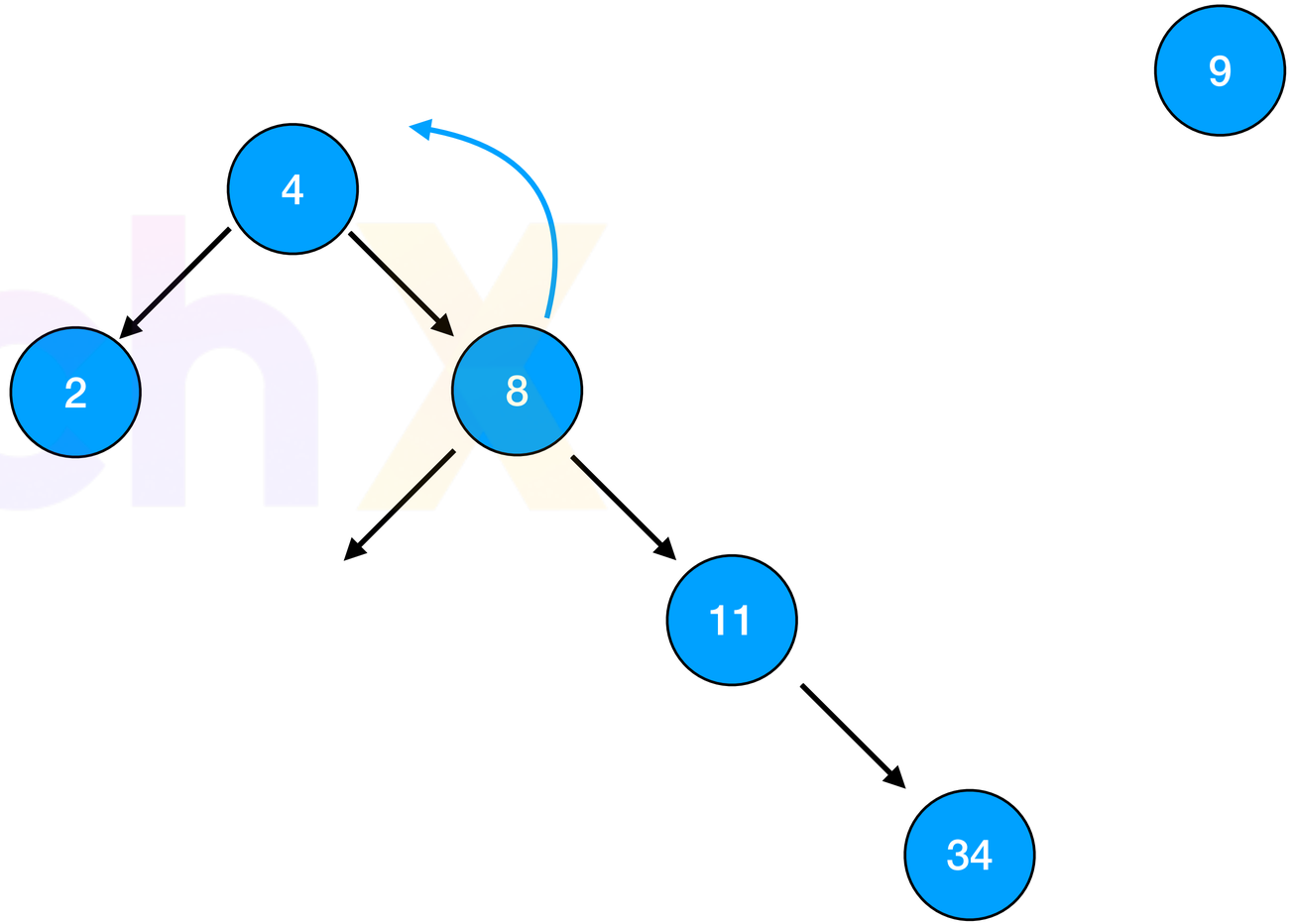


name

price

Category

price = [2 , 4 , 8 , 11 , 34 , 9]



```
class AVLNode:
    def __init__(self, key):
        self.value = key
        self.left = None
        self.right = None
        self.height = 1
```

```
class AVLTree :
    def insert (self, root, key):
        if root == None:
            return AVLNode(key)
        elif key < root.value:
            root.left = self.insert(root.left, key)
        else
            root.right = self.insert(root.right, key)
```

Height of the Node

Balance Factor

AVL tree rotations

```
class AVLTree :
    def insert (self, root, key):
        if root == None:
            return AVLNode(key)
        elif key < root.value:
            root.left = self.insert(root.left, key)
        else
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.get_height(root.left),
                              self.get_height(root.right))
        .
```

Height of the Node

Balance Factor

AVL tree rotations

```
def get_height(self, node):
    if not node:
        return 0
    return node.height
```

```
class AVLTree :
    def insert (self, root, key):
        if root == None:
            return AVLNode(key)
        elif key < root.value:
            root.left = self.insert(root.left, key)
        else
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.get_height(root.left),
        .                     self.get_height(root.right))
```

Balance Factor

AVL tree rotations

```
def get_height(self, node):
    if not node:
        return 0
    return node.height
```

```
class AVLTree :
    def insert (self, root, key):
        if root == None:
            return AVLNode(key)
        elif key < root.value:
            root.left = self.insert(root.left, key)
        else
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.get_height(root.left),
        .                     self.get_height(root.right))

        balance_factor = self.get_balance(root)
```

Balance Factor

AVL tree rotations

```
def get_balance(self, node):
    if not node:
        return 0
    return self.get_height(node.left) - self.get_height(node.right)
```

```
class AVLTree :
    def insert (self, root, key):
        if root == None:
            return AVLNode(key)
        elif key < root.value:
            root.left = self.insert(root.left, key)
        else
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.get_height(root.left),
        .                     self.get_height(root.right))

        balance_factor = self.get_balance(root)
```

AVL tree rotations

```
def get_balance(self, node):
    if not node:
        return 0
    return self.get_height(node.left) - self.get_height(node.right)
```

```
class AVLTree :
    def insert (self, root, key):
        if root == None:
            return AVLNode(key)
        elif key < root.value:
            root.left = self.insert(root.left, key)
        else
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.get_height(root.left),
        .                     self.get_height(root.right))

        balance_factor = self.get_balance(root)
```

AVL tree rotations