# Advanced Tree Structures – 2– 4 Trees

## Introduction

In the previous set of notes we introduced the *multi-way tree* and specifically the variant known as an *m-way search tree.* In this set of notes we will examine a special variant of *m-way search trees* that has important applications in the data structures area.

## 2-4 Trees

If there is a maximum value *m* placed on the number of children that a given node may have, the tree is referred to as an *m-way tree*. In this section we will focus on a common variant of the *m-way tree* known as a *2-3-4 tree* or more commonly as a *2-4 tree.*

**2-4 Tree**

A 2-4 tree is an m-way search tree *T* in which an ordering is imposed on the set of keys which reside in each node such that:

1. Each node has a maximum of 4 children and between 1 and 3 keys.
2. The keys in each node appear in ascending order.
3. The keys in the first *i* children are smaller than the *i*th key.
4. The keys in the last *m-1* children are larger than the *i*th key.
5. All external nodes have the same depth.

In the above definition, rule 1 defines a *size property* for the 2-4 tree; rules 2, 3, and 4 define the *ordering property* (which identifies the tree as a search tree), and rule 5 defines a *depth property* which determines the balance of a 2-4 tree. This depth property ensures that the height of a 2-4 tree containing *n* key values is $\theta(\log_2 n)$. Figure 1 shows a 2-4 tree containing 13 key values (items) with a height of three (not counting the external nodes).

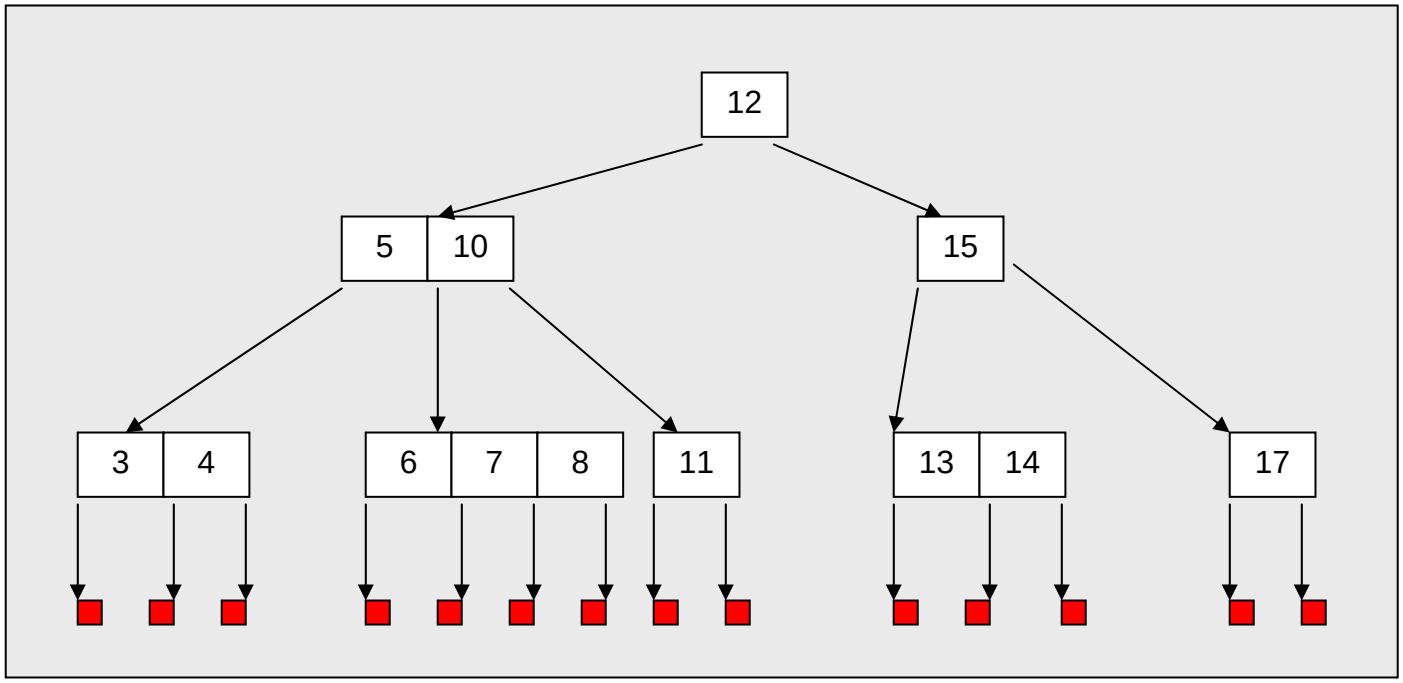



Figure 1. 2-4 tree containing 13 key values.

## Insertion into a 2-4 Tree

As with the other types of search trees, the insertion of a new item (*k, x)*, where *k* is the key value of item *x,* into a 2-4 tree begins with a search for the key value *k*. Assuming that the item does not already exist, the search will terminate unsuccessfully at an external node, let's call it *z*. If *v* is the parent of this external node *z*, then the new item is inserted into node *v* and a new child is added to *v*. Let's call this new child *w*, and we know that *w* is an external node. While this insertion technique clearly preserves the depth property of the 2-4 tree it may well violate the size property. The problem is that node *v* may already have four children and thus be a *4-node*. Insertion of a new node in this manner would cause node *v* to become a 5-node and thus violate the size property.

Any time an insertion occurs in node which is already a 4-node an *overflow* occurs and resolution of the overflow must occur to restore the properties of the 2-4 tree. Resolution of the overflow is done via a *splitting* operation. Recall that we saw this operation when we examined the general *m-way search tree*.

The definition of the split operation for a 2-4 tree is shown in Figure 2. The splitting operation is shown in Figures 3, 4 and 5.

Node *v* is a 4-node (has four children).

1. Replace node *v* with two nodes *v1* and *v2* where"

   - *v1* is a 3-node with children $v_1$, $v_2$, $v_3$ storing keys *k1* and *k2*.
   - v2 *is a 2-node* with children $v_4$, $v_5$, storing key $k_4$.

2. If *v* was the root of the tree, create a new root node *u*; otherwise, let *u* be the parent of *v*.

3. Insert key *k3* into *u* and make *v1* and *v2* children of *u*, so that if *v* was the *i*th child of *u*, then *v1* and *v2* become children *i* and *i+1* of node *u* respectively.

Figure 2 - Insertion Rules for Overflowing Node.

.

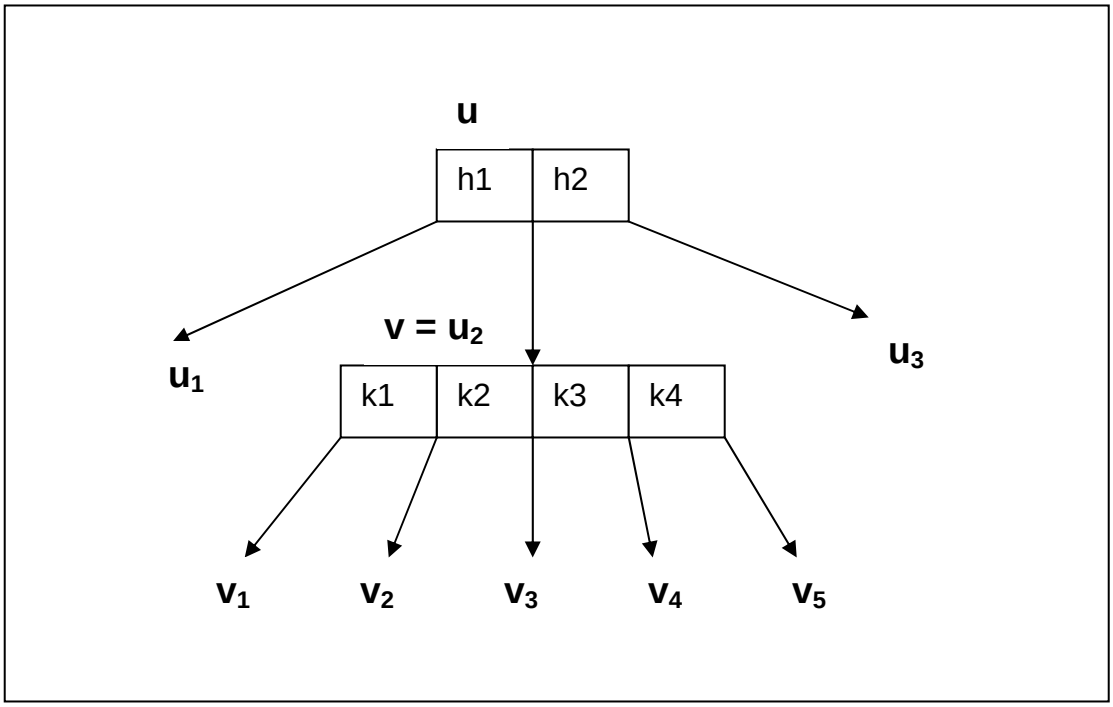



Figure 3 – Node Splitting. Overflow creates a 5-node at *v*.

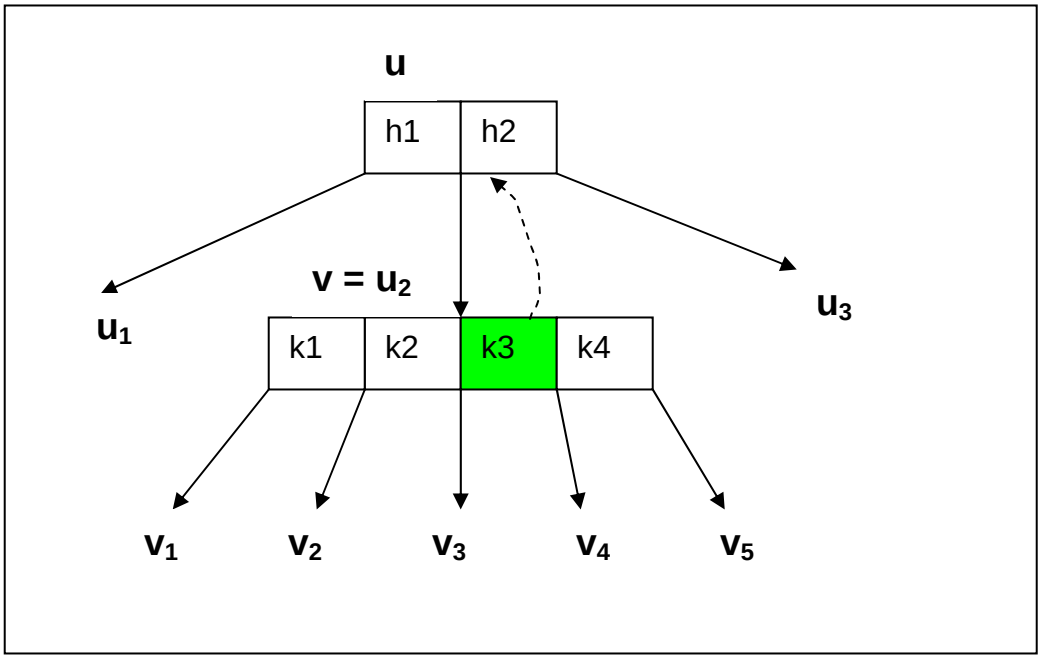



Figure 4 – Node Splitting. Move *k3* into *u*, effectively splitting *v*.

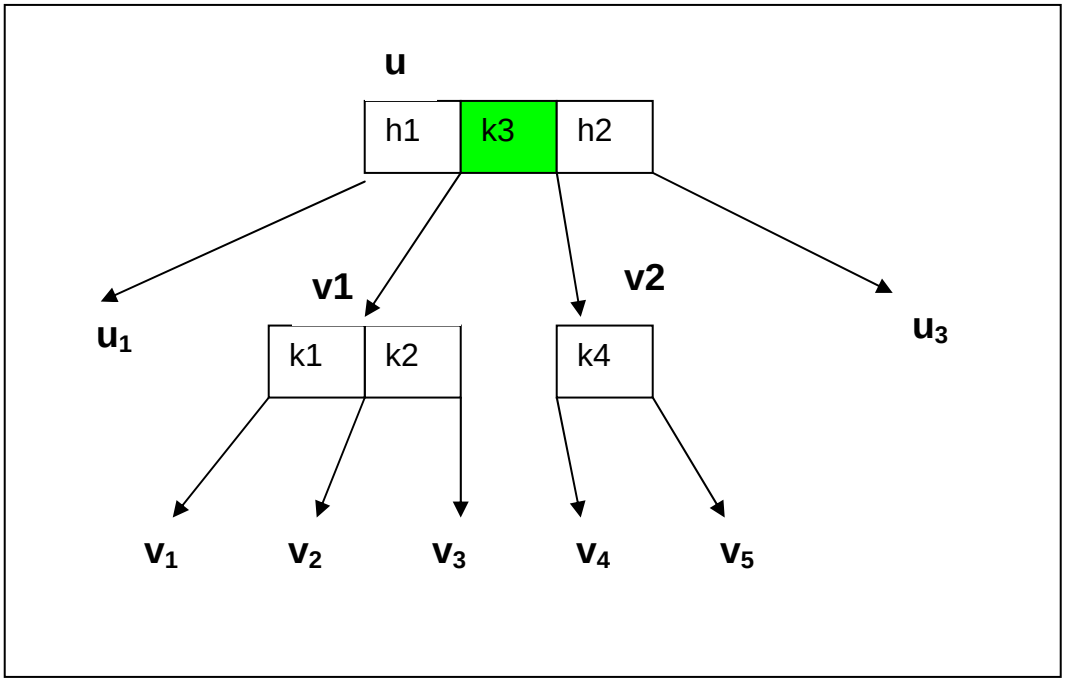


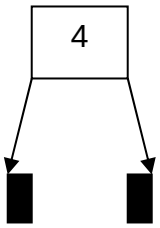

Figure 5 – Node Splitting. Node *v* is split creating two nodes *v1* and *v2*.

The following example illustrates a sequence of insertions into an initially empty 2-4 tree. Overflow causing splitting as well as the creation of a new root node are illustrated in this example.

Example

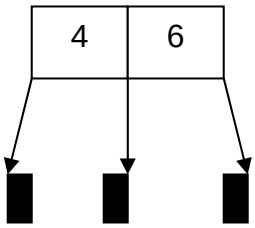
(b)

Sequence of insertions is: 4, 6, 12, 15, 3, 5, 10, and 8.



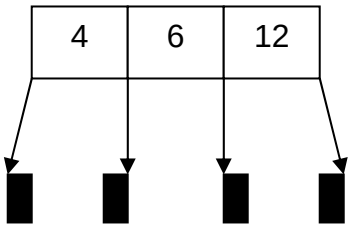

(a)

Insertion of 4



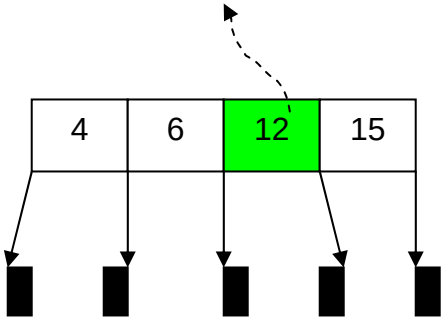

(b)

Insertion of 6



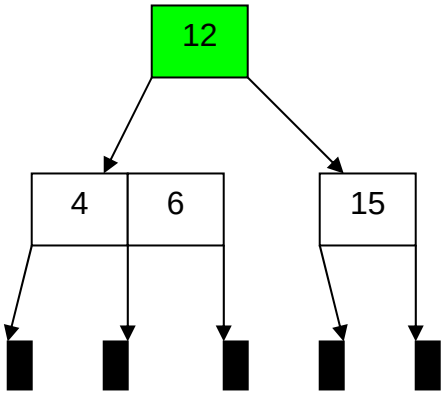

(c)

Insertion of 12



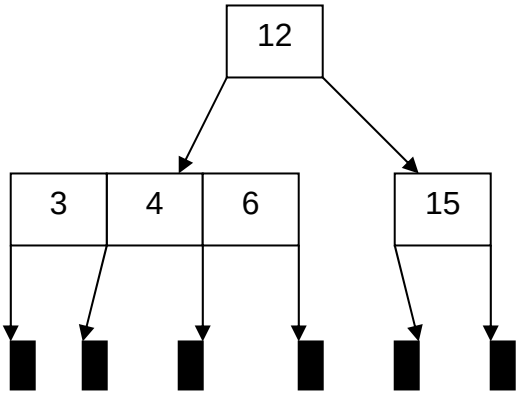

(d)

Insertion of 15 causes overflow condition



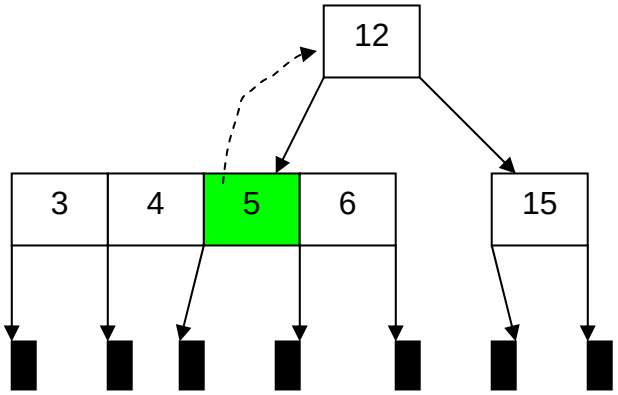

(e)

Overflowing node is split causing the creation of a new root node



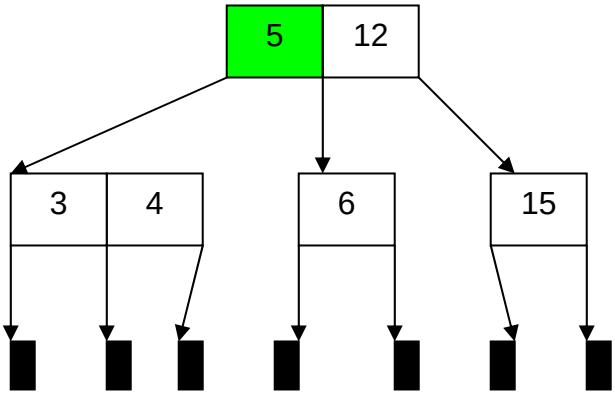

(f)

Insertion of 3



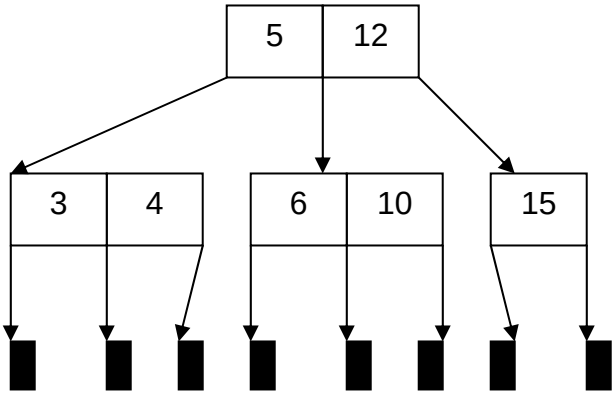

(g)

Insertion of 5 causes overflow and splitting.



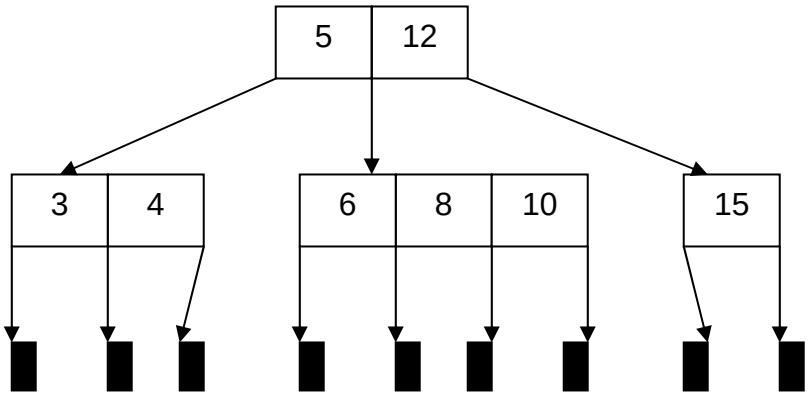

(h)

After splitting overflowing 5-node




(h)

Insertion of 10




(i)

Insertion of 8

## Cascade Splitting

Inserting a new item into a node which already contains three items causes the splitting operation that we have detailed above. The problem with this operation is that there is no guarantee that the parent of the split node will have room for the overflowing item. Therefore, splitting the child of a parent may well lead to the splitting of the parent, which may in turn lead to the splitting of the grandparent and so on. The splitting operation may cascade all the way to the root of the tree, which in turn may be split. Since we are assured that the height of a 2-4 tree containing *n* items has a $\log_2 n$ bound, then the splitting operation is bounded by this height and therefore the total time required to perform an insertion is $O(\log_2 n)$. The sequence of 2-4 trees illustrated in Figure 6 shows cascade splitting (in the interest of room, the external nodes are not shown in this figure).

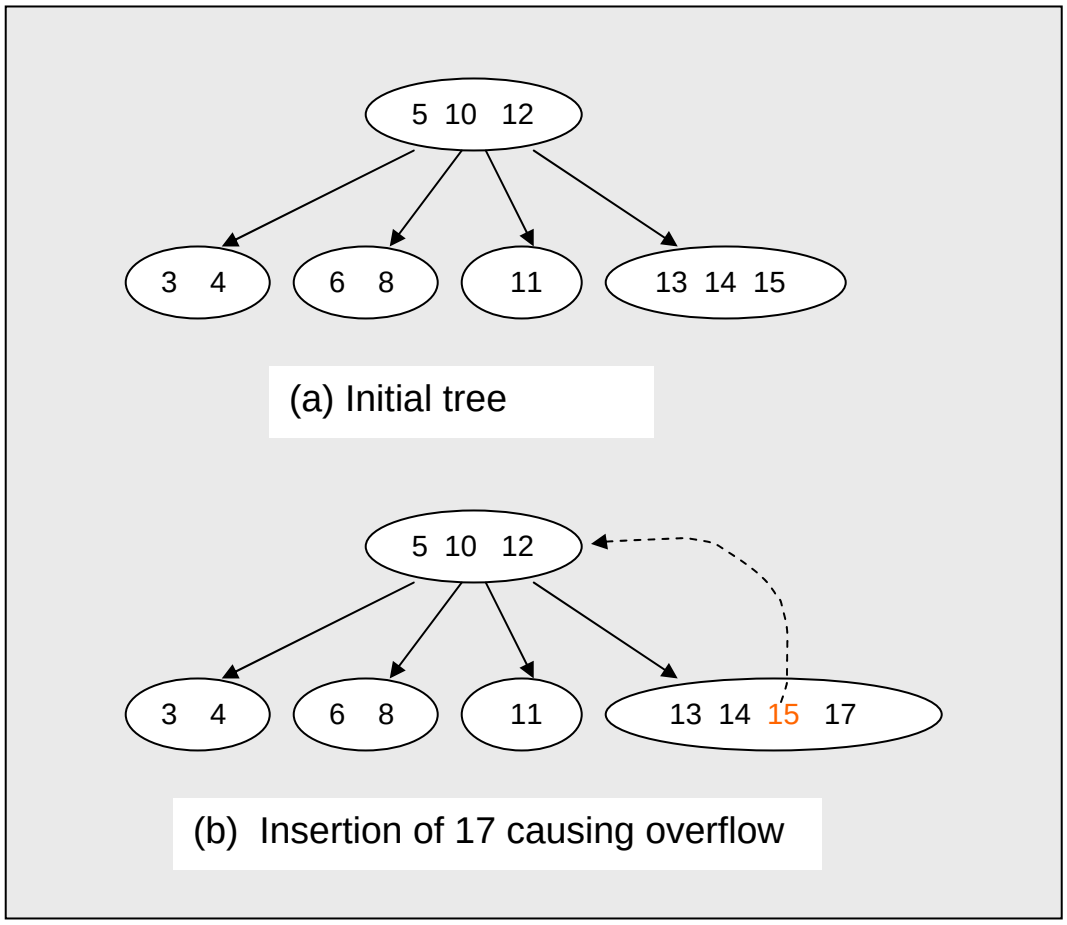



Figure 6 – Cascade Splitting. (a) Initial 2-4 tree. (b) Insertion causing splitting.

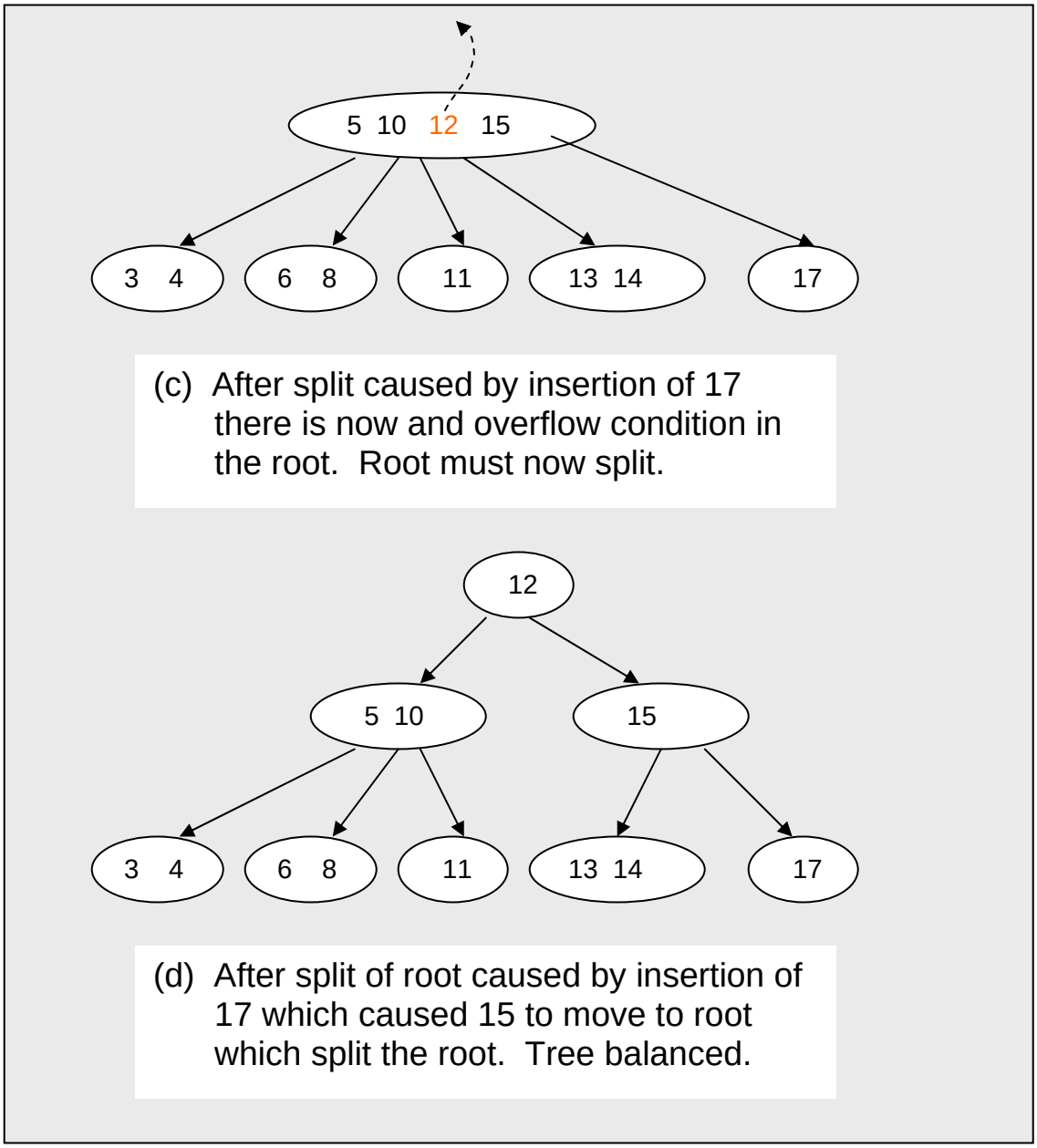



Figure 6 continued – (c) Insertion causes splitting in both parent and grandparent (root in this case). (d) Balanced 2-4 tree after splitting root node in final split of a cascade of splits.

## Deletion from a 2-4 Tree

Deletion of an item from a 2-4 tree proceeds in much the same way as the deletion from a BST. First the item to be deleted must be found in the tree which means that a search will be performed in the tree. Deleting any item from the tree can be reduced to the trivial case where the item to be deleted is contained in a node that has only external nodes as children. Suppose that we want to remove an item with key $k$ which is stored as the $i$th item $(k_i, x_i)$ in node

*z*, where *z* has only internal nodes as children. In order to reduce this deletion to the trivial case, we need to swap the item $(k_i, x_i)$ with an appropriate item that is stored at a node *v* which has only external nodes as children. To do this we need to:

1. Find the right-most internal node *v* in the subtree rooted at the *i*th child of *z*, noting that the children of node *v* are all external nodes.
2. Swap the item $(k_i, x_i)$ at *z* with the last item of *v*.

Deleting a node in this fashion from a 2-4 tree guarantees that the depth property will be preserved since we always delete an external node child from a node *v* with only external-node children. However, in removing such an external node, the technique may violate the size property at node *v*. Indeed, if node *v* was previously a 2-node, then it becomes a 1-node with no items after the removal and such a node is not allowed in a 2-4 tree. This type of violation of the size property is called an *underflow* at node *v*. To resolve an underflow, we check whether an immediate sibling of *v* is a 3-node or 4-node. If we find such a sibling *w*, then a *transfer* operation is performed in which, a child of *w* is moved to *v*, a key of *w* is moved to the parent of *v* and *w* (let's call this parent *u*), and a key of *u* is moved to *v*.

Transfer Operation:

Underflow occurs at node *v* with parent *u*.

1. Find a sibling *w* of *v* that is a 3-node or 4-node.
2. Move a child of *w* to *v*.
3. Move a key of *w* to *u*.
4. Move a key of *u* to *v.*

If node *v* has only one sibling, or if both immediate siblings of *v* are 2-nodes, then a *fusion* operation must be performed, in which node *v* is merged with a sibling creating a new node *v′*, and a key from the parent *u* is moved to *v′*.

Fusion Operation:

Underflow occurs at node *v* with parent *u*.
Node *v* has only one sibling or both immediate siblings are 2-nodes.

1. Merge *v* with a sibling to create a new node *v′*.
2. Move a key of *u* to *v′*.

Figure 7 illustrates a transfer operation. Figure 8 illustrates a fusion operation.

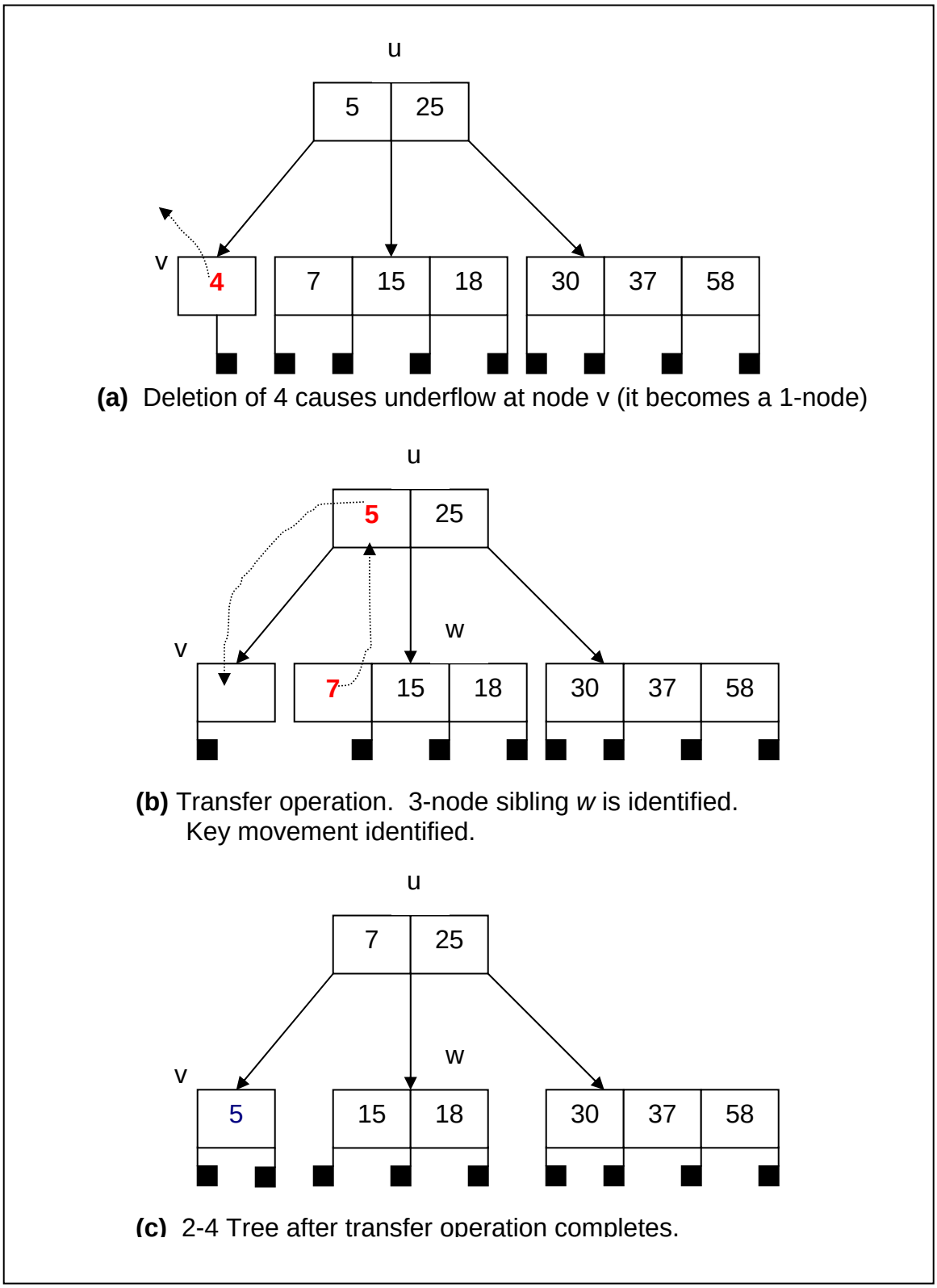



Figure 7 – Transfer operation illustrated. (a) Initial tree showing deletion of node containing 4 which causes underflow. (b) Identification of sibling *w* and movement of key values in the transfer operation. (c) Final 2-4 tree after transfer operation is completed.

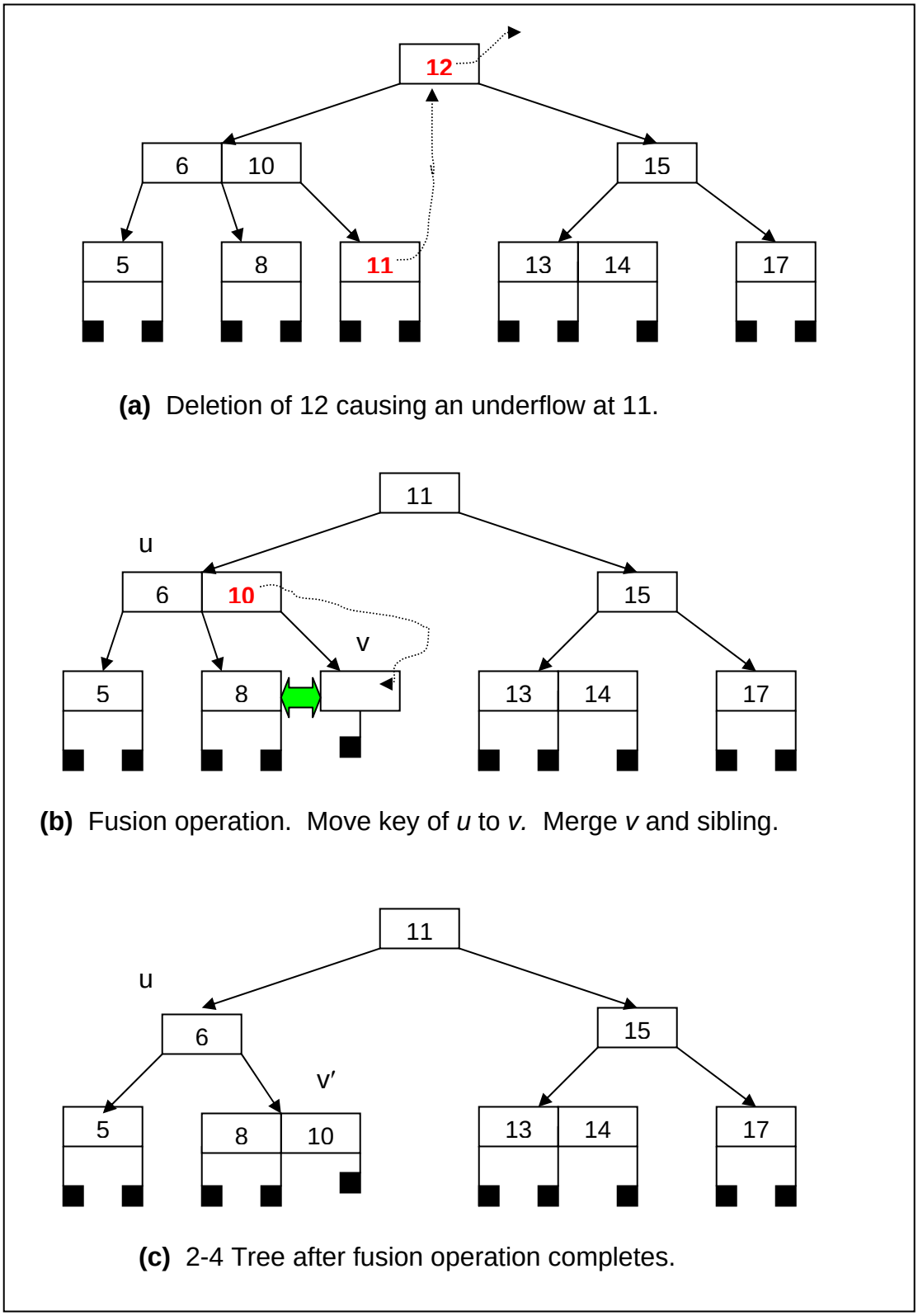



Figure 8 – Fusion operation illustrated. (a) Initial 2-4 tree with deletion of root node 12 causing underflow at node 11 which takes the place of 12 at the root. (b) The fusion operation. Moving a key of *u* to *v* and the merging of *v* and its sibling. (c) Final 2-4 tree after the fusion operation has completed.

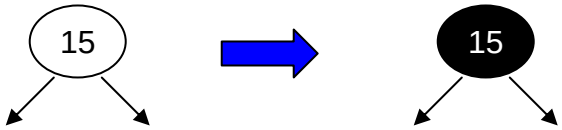
A fusion operation at some node *v* may cause a new underflow to occur at the parent *u* of *v*, which will in turn trigger a transfer or fusion operation at *u*. Thus, as was the case with insertion which could cause a cascading split to occur, so too with deletion there may occur a cascade of transfer and fusion operations.

You'll be able to practice these operations in the next set of homework problems!

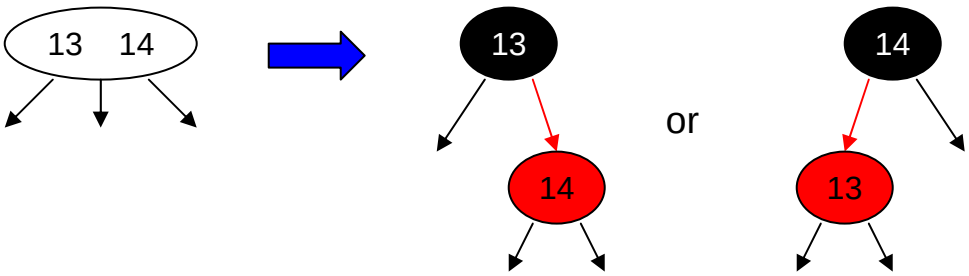
## Summary

There is an interesting correspondence between 2-4 trees and red-black trees. Given a red-black tree, the corresponding 2-4 tree can be constructed by merging every red node *v* into its parent and storing the item from *v* at its parent. Conversely, any 2-4 tree can be transformed into its corresponding red-black tree by coloring each node black and performing the following transformation for each internal node *v*:

- If *v* is a 2-node, the keep the black children of *v* as is.




- If *v* is a 3-node, then create a new red node *w*, give *v*'s first two black children to *w*, and make *w* and *v*'s third child be the two children of *v*.




- If *v* is a 4-node, then create two new red nodes *w* and *z*, give *v*'s first two black children to *w*, give *v*'s last two black children to *z*, and make *w* and *z* be the two children of *v*.