CMPT 225: Data Structures & Programming – Unit 25 – (2, 4) Trees

Dr. Jack Thomas

Simon Fraser University

Spring 2021

Today’s Topics

- Multi-Way Search Trees
- Multi-Way In-Order Traversals and Searching
- (2, 4) Trees
- Overflows and Underflows
- Analysis

We’ve Had Non-Linear and Non-Positional

- Now it’s time for **Non-Binary**.
- We’re not just talking about Trees that can have more than two kids (we already had that with **general Trees**).
- These are Trees with **more than one entry per node**, and a **number of children based on the number of entries**, all to create a new type of **search tree**.

The Multi-Way Search Tree

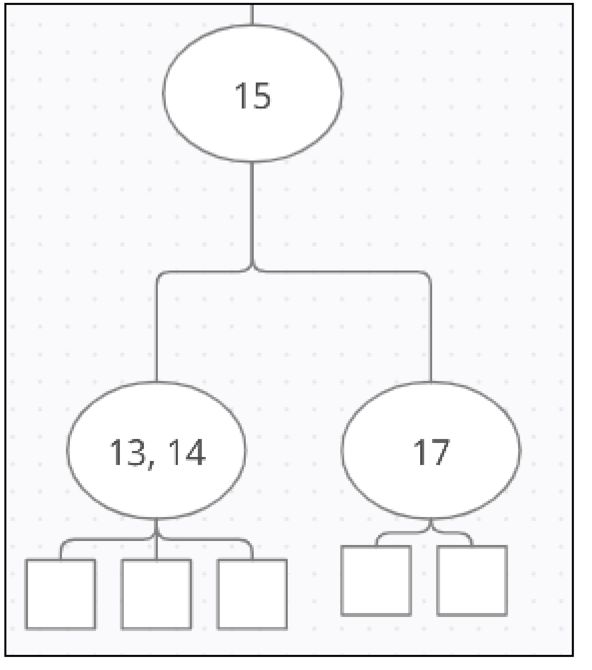
- A Tree where each internal node may have many children and many entries is called a **Multi-Way Tree**.
- By enforcing an **ordering** on the keys stored in those entries, we can create an **alternative data structure** to the **Binary Search Tree**, called the **Multi-Way Search Tree**.

Defining the Multi-Way Search Tree

- Let v be a node of an **ordered tree**. We say that v is a d-node if v has d children. We define a **Multi-Way Search Tree** to be an ordered tree T obeys the **following three rules**.

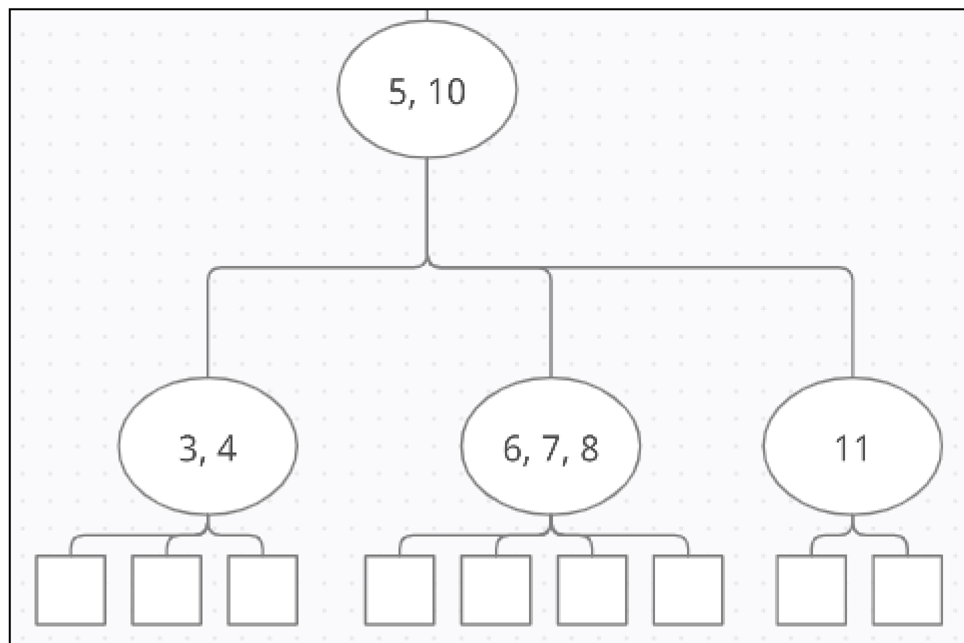
1. Minimum Child Requirement

- Each internal node of T has **at least two children**. That is, each internal node is a d-node such that d >= 2.
- These children may be other internal nodes, or **blank external nodes** (the same sort of blank externals we used with the Binary Search Tree).



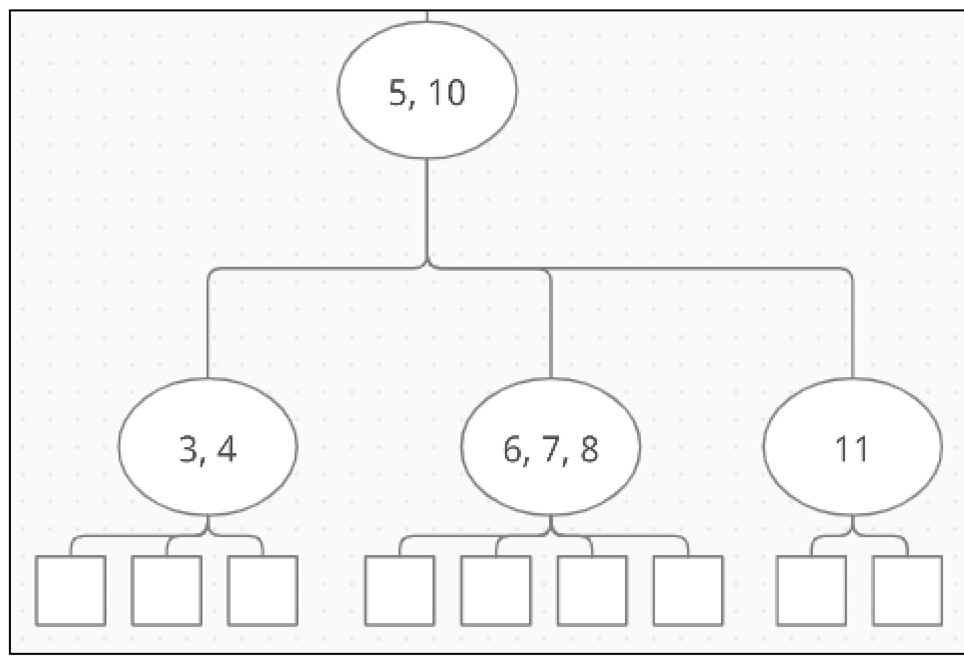
2. Key-Child Correlation

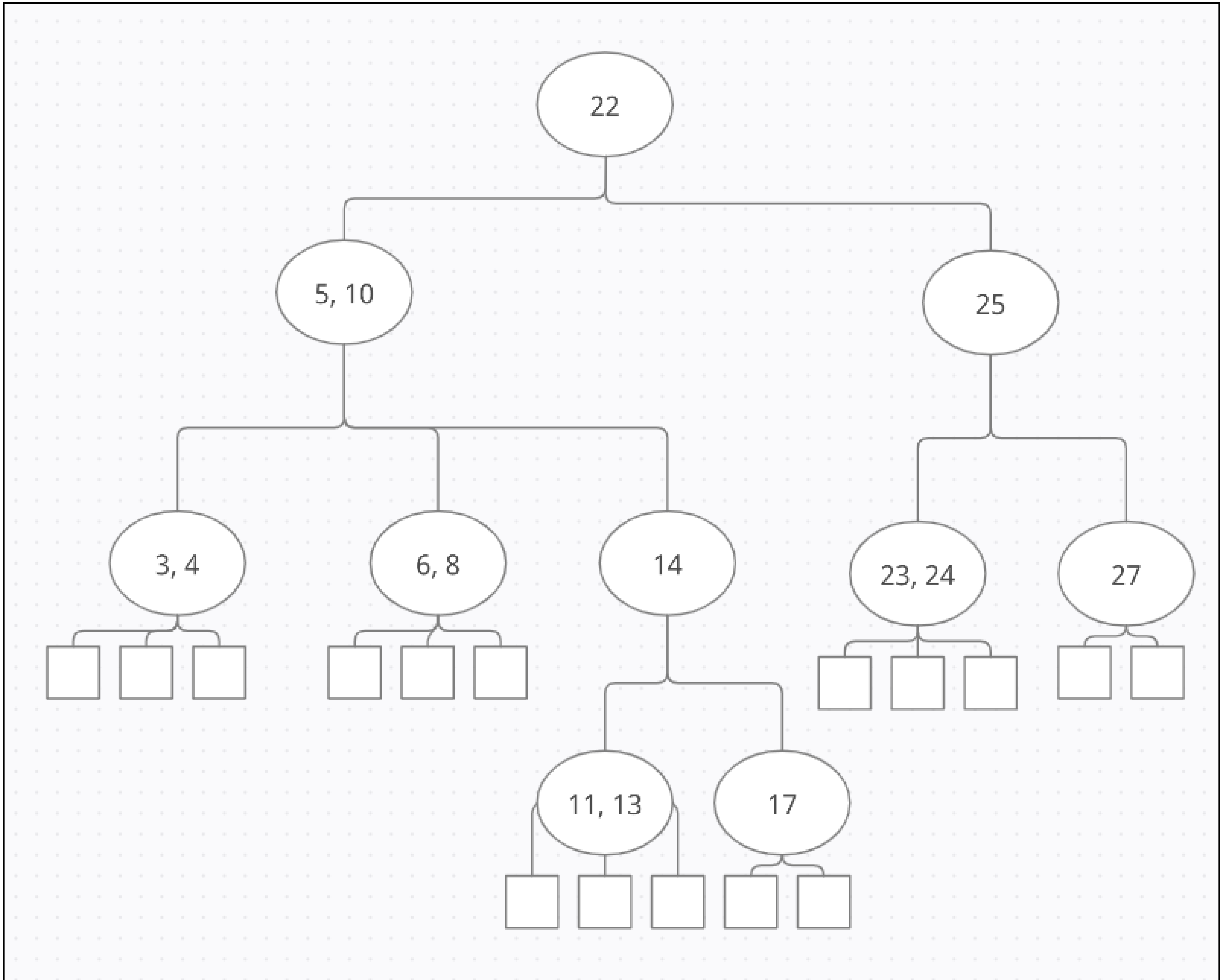
- Each internal d-node v of T with children $v_1,\ldots,v_d$ stores an **ordered set of d – 1 key-value entries** $(k_1, x_1), \ldots, (k_{d-1}, x_{d-1})$, where $k_1 <= \ldots <= k_{d-1}$.
- For example, here we have a node with **two keys stored in order** (5 and 10) and **three internal children**.



3. Interleaved Ordering

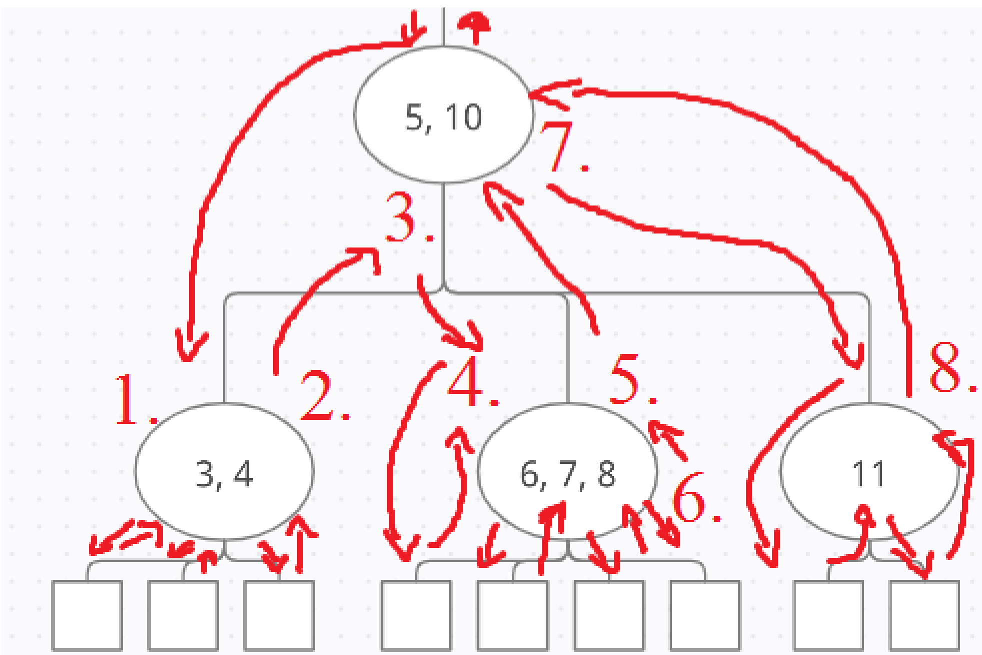
- Let us conventionally define k_0 = -inf and k_d = +inf. **For each entry (k, x)** stored at a node in the **subtree of v rooted at v_i**, i =1,…,d, we have $k_{i-1} <= k <= k_i$.
- The **leftmost child's keys** are smaller than 5, the **middle child's keys** are between 5 and 10, and the **rightmost child's key** is greater than 10.




22
5, 10
25
3, 4
6, 8
14
23, 24
27
11, 13
17

The Multi-Way In-Order Traversal

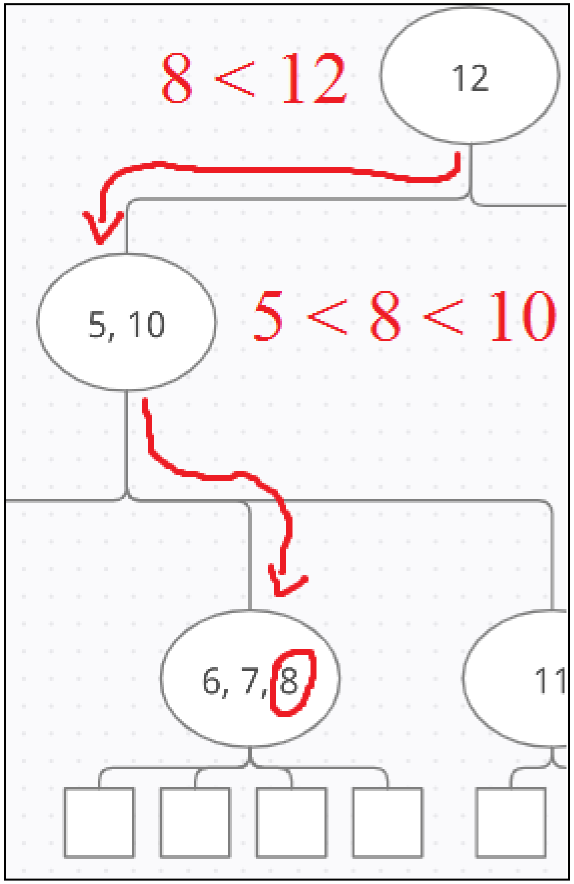
- A Multi-Way Search Tree can be **traversed in order** in the same way that a Binary Search Tree can, just by opening up the algorithm from “go left, this node, go right, go back” to a loop through the children and stored entries from left to right.



- Does covering this thing in red arrows help? I hope it helps.

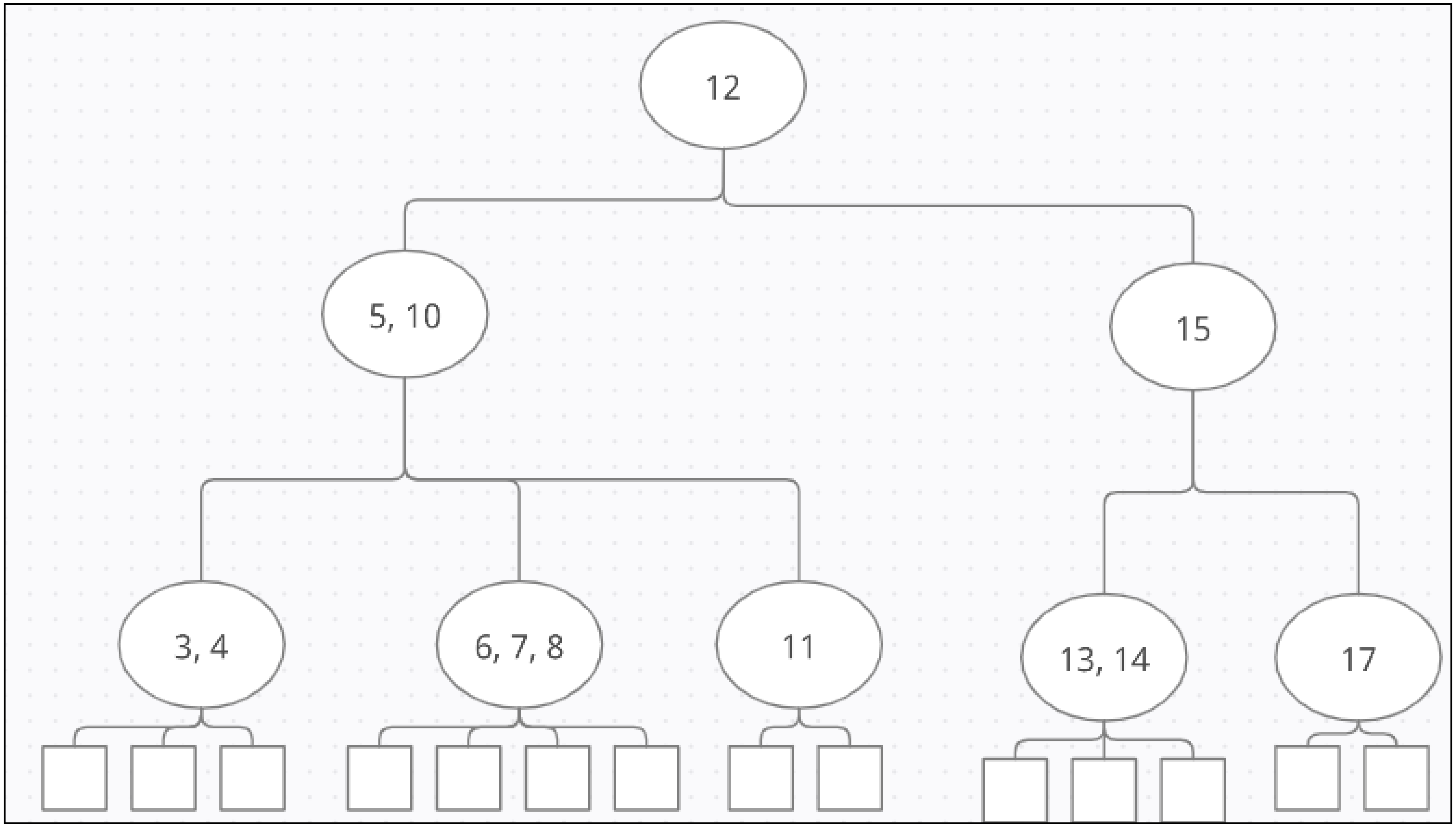
Multi-Way Searching

- Thanks to the way the keys are ordered, **searching for a given key** is quite simple.
- If the search key is greater than all keys at the current node, **take the rightmost link**.
- If it's smaller, **take the leftmost link**.
- Otherwise, **take the link between the keys** that're smaller and larger than the search key.
- If you find an external node, **the key isn't in here**.



(2, 4) Trees

- Sometimes called **2-3-4 Trees**, these are Multi-Way Search Trees with two additional properties:
  - **The Node Size Property**, where every internal node has a maximum of 4 children.
  - **The Depth Property**, where every external node has the same depth.
- Internal nodes are either **2-nodes, 3-nodes, or 4-nodes**, based on the number of children.


12
5, 10
15
3, 4
6, 7, 8
11
13, 14
17

This Is A Lot Of Rules, Why Are We Doing This Again?

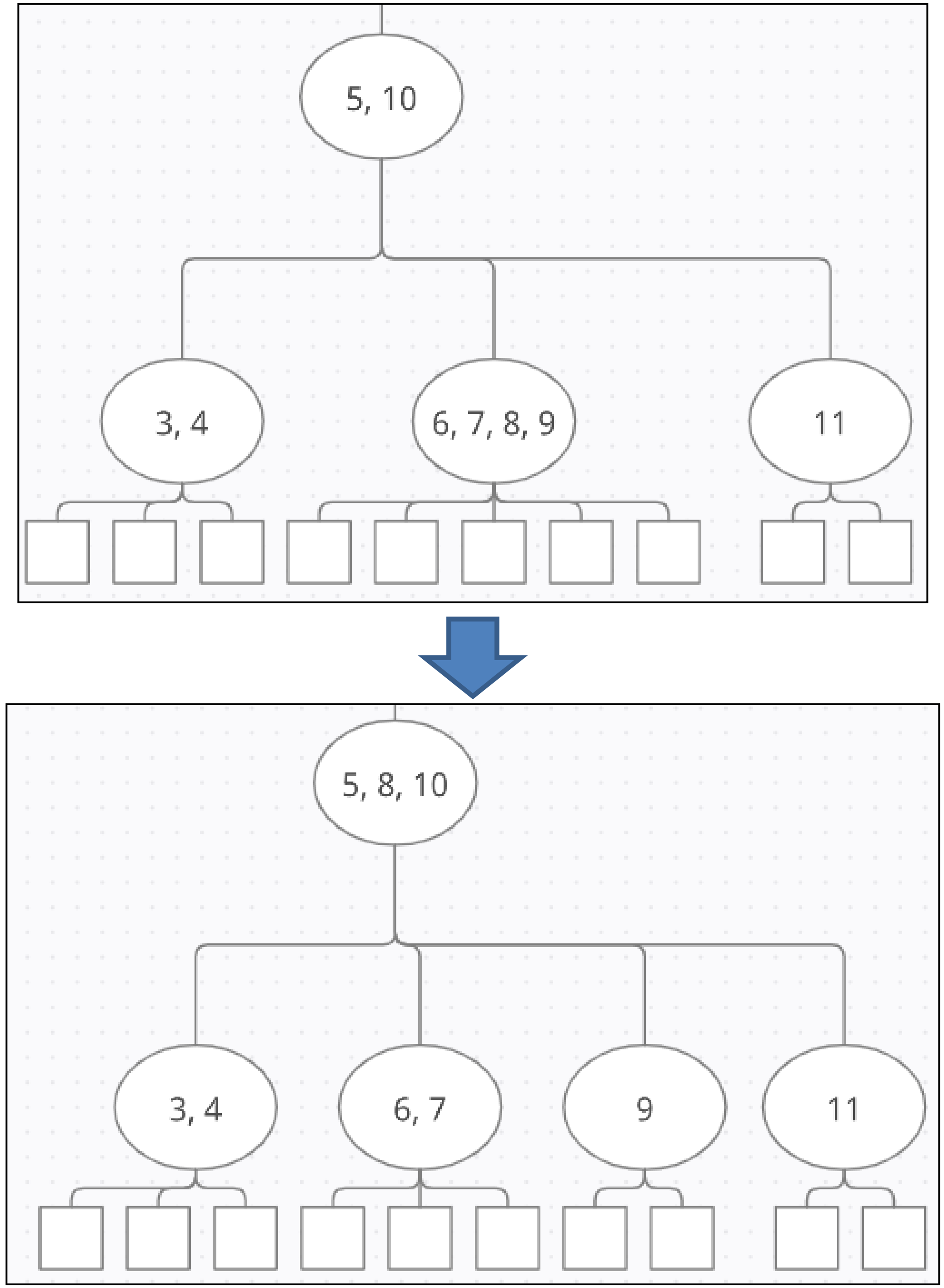
- Where an **AVL-BST** gave us height-bounded, O(log n) methods through **complex functions** like restructuring, **(2,4) Trees** give us the same efficiency through their **complex structure**.
- Requiring all external nodes to be the same depth while also capping internal node children to between 2 and 4 will indirectly cause the **completeness property**, as even the least compact valid Tree would simply be a Complete Binary Tree, which we know has **h = log n**.

Building a (2, 4) Tree

- Like with a BST, **insertions** into a (2, 4) Tree start with a Multi-Way Search to bring us to the **external node** where the new key would fit.
- Instead of adding the new entry to the external node (violating the rule about external node depths), we just **add it to the internal parent’s set of keys**, then create a new external node and link to interleave between the parent’s expanded key set.

Dealing With Overflows

- If the parent **already has four children**, then an insertion which bumps that up to five causes an **overflow**.
- This leads to a **split operation**, where a node is replaced with two nodes and the four keys are divided between them and their parent.
- Keys 1 and 2 go to one new node, key 3 goes to the parent (to sit between them), and key 4 goes to the next new node.


5, 10
3, 4
6, 7, 8, 9
11
5, 8, 10
3, 4
6, 7
9
11

What If That Causes The Parent to Overflow?

- Then **the split operation repeats** for the parent as well.
- This can **propagate all the way up to the root**, which is how the height of the tree increases – by the root splitting in two and creating a new root above the two halves of what was previously the root.

Deletion

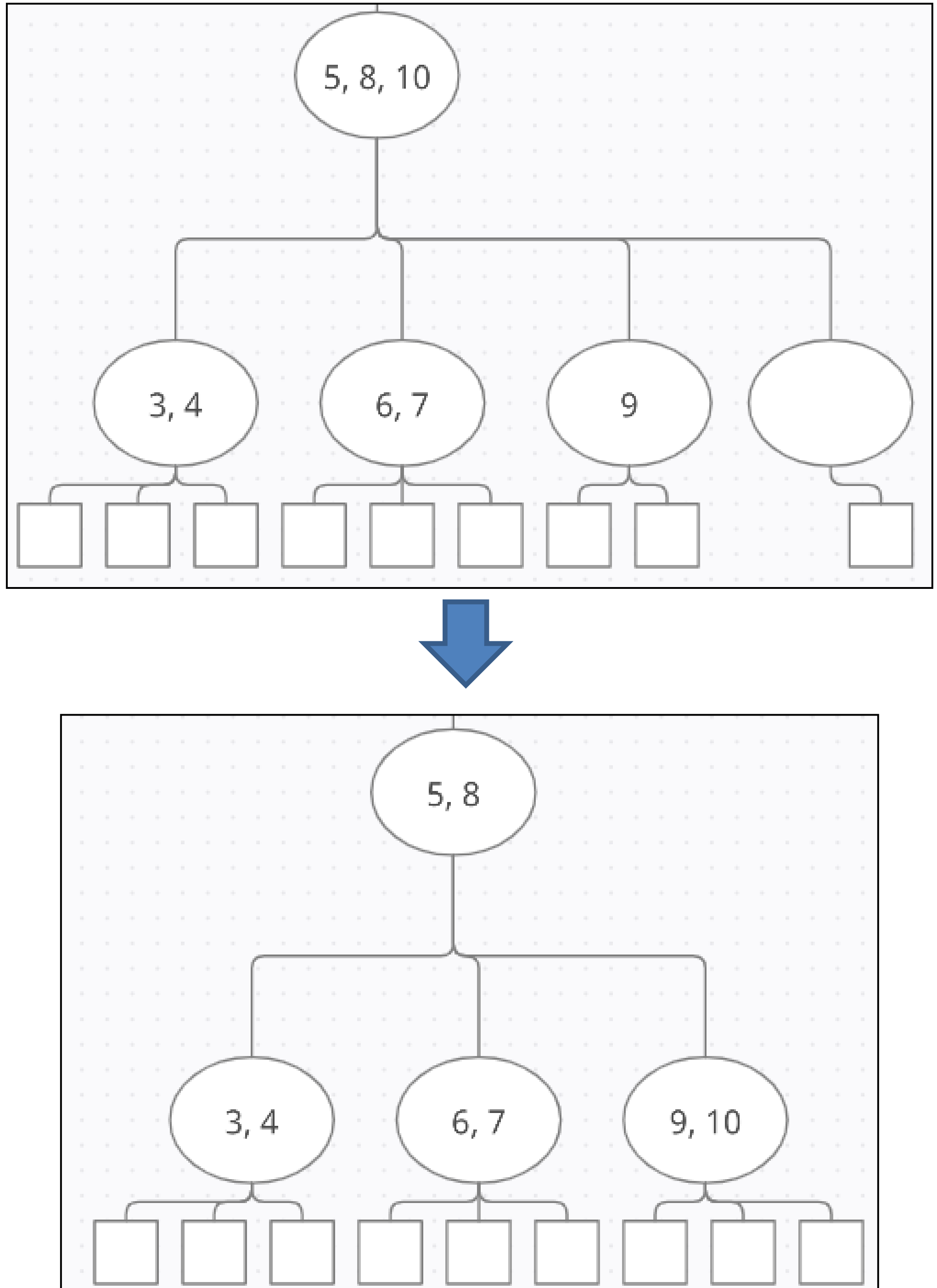
- Also begins with a search. If **the key is found in a node with only external children**, removing it is easy – just delete it from the set and delete one of the external nodes.
- If **the key is in a node with internal children**, instead swap in a key from one of the children based on the in-order succession (i.e. the largest child).
- This process must **repeat for the child node the key was taken from** and its children, until a child with external nodes is reached, at which point you can do the easy deletion and removing an external node from the first case.

Handling Underflows

- If the node with external children from who a key is deleted is now a 1-node (one external child, no keys), this causes an **underflow**.
- Handling an underflow for a node will involve its parent and adjacent siblings, and falls into **one of two cases depending on the size of their siblings**.

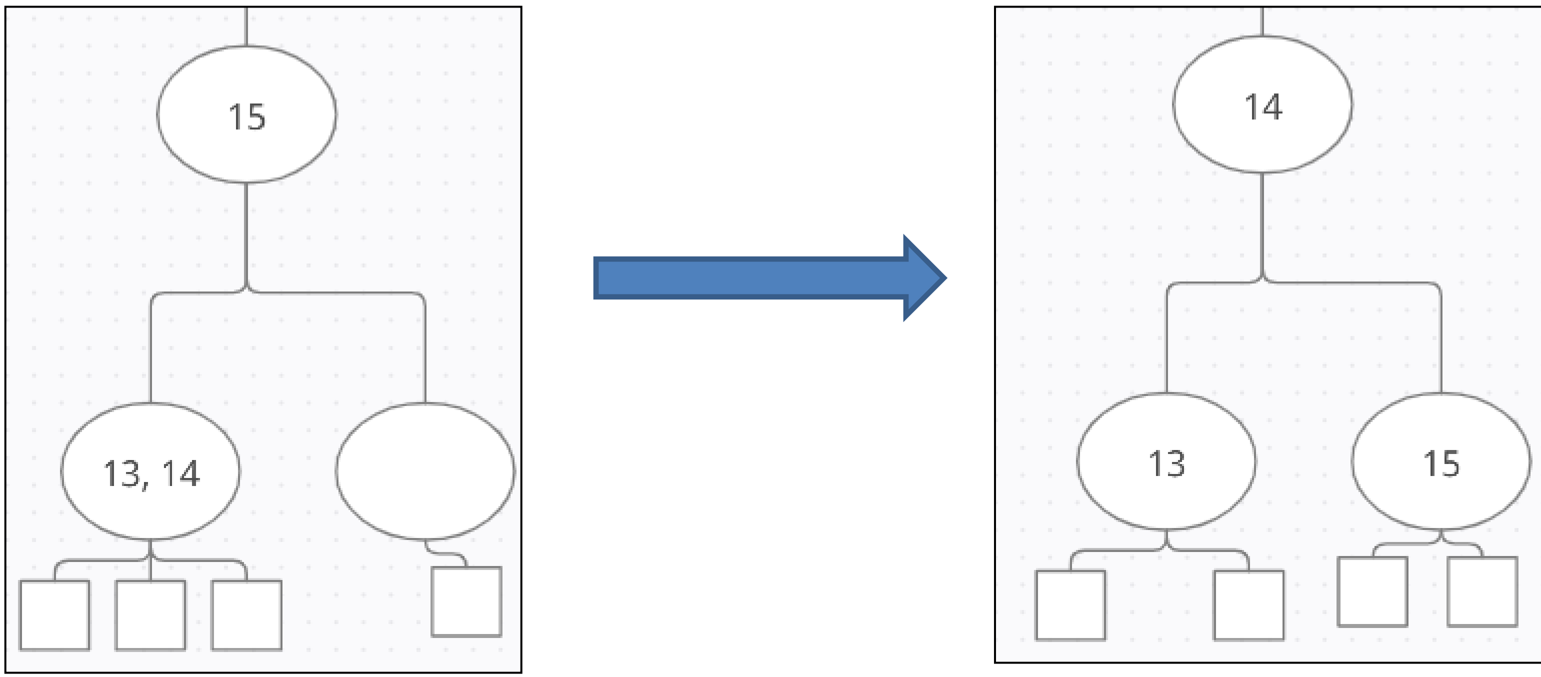
Case 1: Fuse With a Sibling

- If **both siblings** of the underflowing node are **2-nodes**, we perform a **fusion operation**.
- We **merge the node with an adjacent sibling into a single new node**, and then take in the key from their parent that was previously between the two of them.
- Note that taking that key could cause the **underflow to propagate to the parent**, so it's time to repeat the underflow operation on them!


5, 8, 10
3, 4
6, 7
9
5, 8
3, 4
6, 7
9, 10

Case 2: Transfer From a Sibling

- If either **adjacent sibling** of the underflow node is a **3 or 4-node**, then we can **transfer the key from the parent** that's between those two nodes and give it to the underflowing node.
- We then **give the parent a key from the sibling** to maintain the ordering, causing no underflow, which prevents it from propagating.


15
13, 14
14
13
15

Recap – 2 Tree 4 Me

- A **Multi-Way Search Tree** is an ordered Tree that allows multiple entries and children per node, according to a special correlation
- We can find keys through **Multi-Way Search**, and a special version of **In-Order Traversals**.
- The **(2, 4) Tree** adds additional constraints to ensure the Tree's height is log n.
- This requires special rules for handling **overflows on insertions** and **underflows on removals**.