(2,4) Trees
Multi-Way Search Trees

A multi-way search tree is an ordered tree such that:

- Each internal node has at least two children and stores \( d - 1 \) key-element items \( (k_i, o_i) \), where \( d \) is the number of children.
- For a node with children \( v_1, v_2, \ldots, v_d \) storing keys \( k_1, k_2, \ldots, k_{d-1} \):
  - keys in the subtree of \( v_1 \) are less than \( k_1 \)
  - keys in the subtree of \( v_i \) are between \( k_{i-1} \) and \( k_i \) \((i = 2, \ldots, d - 1)\)
  - keys in the subtree of \( v_d \) are greater than \( k_{d-1} \)
- The leaves store no items and serve as placeholders.
Multi-Way Inorder Traversal

We can extend the notion of inorder traversal from binary trees to multi-way search trees.

Namely, we visit item \((k_i, o_i)\) of node \(v\) between the recursive traversals of the subtrees of \(v\) rooted at children \(v_i\) and \(v_{i+1}\).

An inorder traversal of a multi-way search tree visits the keys in increasing order.
Multi-Way Searching

- Similar to search in a binary search tree
- Each internal node with children $v_1, v_2, \ldots, v_d$ and keys $k_1, k_2, \ldots, k_d$:
  - $k = k_i (i = 1, \ldots, d - 1)$: the search terminates successfully
  - $k < k_1$: we continue the search in child $v_1$
  - $k_{i-1} < k < k_i (i = 2, \ldots, d - 1)$: we continue the search in child $v_i$
  - $k > k_{d-1}$: we continue the search in child $v_d$
- Reaching an external node terminates the search unsuccessfully
- Example: search for 30

![Diagram](image-url)
A (2,4) tree (also called 2-4 tree or 2-3-4 tree) is a multi-way search with the following properties:
- **Node-Size Property**: every internal node has at most four children
- **Depth Property**: all the external nodes have the same depth

Depending on the number of children, an internal node of a (2,4) tree is called a 2-node, 3-node or 4-node.
Height of a (2,4) Tree

Theorem: A (2,4) tree storing $n$ items has height $O(\log n)$

Proof:
- Let $h$ be the height of a (2,4) tree with $n$ items
- Since there are at least $2^i$ items at depth $i = 0, \ldots, h - 1$ and no items at depth $h$, we have
  $$n \geq 1 + 2 + 4 + \ldots + 2^{h-1} = 2^h - 1$$
- Thus, $h \leq \log (n + 1)$

Searching in a (2,4) tree with $n$ items takes $O(\log n)$ time
Insertion

We insert a new item \((k, o)\) at the parent \(v\) of the leaf reached by searching for \(k\).

- We preserve the depth property but
- We may cause an overflow (i.e., node \(v\) may become a 5-node)

Example: inserting key 30 causes an overflow

```
  10 15 24
 /  \  /  \\
2 8 12 18
|     |   |
2 8 12 30 32 35
```

© 2004 Goodrich, Tamassia

(2,4) Trees
Overflow and Split

We handle an overflow at a 5-node \( v \) with a split operation:

- let \( v_1 \ldots v_5 \) be the children of \( v \) and \( k_1 \ldots k_4 \) be the keys of \( v \)
- node \( v \) is replaced nodes \( v' \) and \( v'' \)
  - \( v' \) is a 3-node with keys \( k_1 k_2 \) and children \( v_1 v_2 v_3 \)
  - \( v'' \) is a 2-node with key \( k_4 \) and children \( v_4 v_5 \)
- key \( k_3 \) is inserted into the parent \( u \) of \( v \) (a new root may be created)

The overflow may propagate to the parent node \( u \)
Let $T$ be a (2,4) tree with $n$ items

- Tree $T$ has $O(\log n)$ height
- Step 1 takes $O(\log n)$ time because we visit $O(\log n)$ nodes
- Step 2 takes $O(1)$ time
- Step 3 takes $O(\log n)$ time because each split takes $O(1)$ time and we perform $O(\log n)$ splits

Thus, an insertion in a (2,4) tree takes $O(\log n)$ time

---

**Algorithm** $\text{insert}(k, o)$

1. We search for key $k$ to locate the insertion node $v$

2. We add the new entry $(k, o)$ at node $v$

3. *while* $\text{overflow}(v)$
   
   *if* $\text{isRoot}(v)$
   
   create a new empty root above $v$
   
   $v \leftarrow \text{split}(v)$
Deletion

- We reduce deletion of an entry to the case where the item is at the node with leaf children.
- Otherwise, we replace the entry with its inorder successor (or, equivalently, with its inorder predecessor) and delete the latter entry.
- Example: to delete key 24, we replace it with 27 (inorder successor).
Underflow and Fusion

- Deleting an entry from a node \( v \) may cause an underflow, where node \( v \) becomes a 1-node with one child and no keys.
- To handle an underflow at node \( v \) with parent \( u \), we consider two cases:
  - **Case 1:** the adjacent siblings of \( v \) are 2-nodes
    - **Fusion operation:** we merge \( v \) with an adjacent sibling \( w \) and move an entry from \( u \) to the merged node \( v' \).
    - After a fusion, the underflow may propagate to the parent \( u \).
Underflow and Transfer

To handle an underflow at node \( v \) with parent \( u \), we consider two cases

- **Case 2:** an adjacent sibling \( w \) of \( v \) is a 3-node or a 4-node
  - **Transfer operation:**
    1. we move a child of \( w \) to \( v \)
    2. we move an item from \( u \) to \( v \)
    3. we move an item from \( w \) to \( u \)
  - After a transfer, no underflow occurs
Analysis of Deletion

Let $T$ be a $(2,4)$ tree with $n$ items
- Tree $T$ has $O(\log n)$ height

In a deletion operation
- We visit $O(\log n)$ nodes to locate the node from which to delete the entry
- We handle an underflow with a series of $O(\log n)$ fusions, followed by at most one transfer
- Each fusion and transfer takes $O(1)$ time

Thus, deleting an item from a $(2,4)$ tree takes $O(\log n)$ time
## Implementing a Dictionary

### Comparison of efficient dictionary implementations

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Insert</th>
<th>Delete</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash Table</td>
<td>1 expected</td>
<td>1 expected</td>
<td>1 expected</td>
<td>no ordered dictionary methods, simple to implement</td>
</tr>
<tr>
<td>AVL Tree</td>
<td>$\log n$ worst case</td>
<td>$\log n$ worst case</td>
<td>$\log n$ worst case</td>
<td>complex to implement</td>
</tr>
<tr>
<td>(2,4) Tree</td>
<td>$\log n$ worst-case</td>
<td>$\log n$ worst-case</td>
<td>$\log n$ worst-case</td>
<td>complex to implement</td>
</tr>
</tbody>
</table>