Multi-Way Search Tree

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 \(k_1, k_2, \ldots, k_d\):
  - keys in the subtree of \(v_i\) are less than \(k_i\)
  - 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

Searching for 30:

Multi-Way Searching

Searching for 30:
Multi-Way Searching

Similar to search in a binary search tree
- A each internal node with children $v_1, v_2, \ldots, v_d$ and keys $k_1, k_2, \ldots, k_{d-1}$:
  - $k = k_i$ ($i = 1, \ldots, d - 1$): the search terminates successfully
  - $k < k_i$: we continue the search in child $v_i$
  - $k_i < k < k_{i+1}$ ($i = 2, \ldots, d - 1$): 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

Multi-Way Searching

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
- As in proper binary trees, there are at least $2^i$ items at depth $i$: $a \geq 2^i - 1$
- Thus, $h \leq \log (a + 1)$
- 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 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
```

```
27   30   32   35
18
```

Overflow and Split

- We handle an overflow at a 5-node \(v\) with a split operation:
  - \(v_1 \ldots v_5\) be the children of \(v\) and \(k_1 \ldots k_5\) be the keys of \(v\).
  - \(v\) is replaced by nodes \(v'\) and \(v''\).
  - \(v'\) is a 3-node with keys \(k_1, k_2\) and children \(v_1, v_2\).
  - \(v''\) is a 2-node with key \(k_5\) 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 \(u\).

Analysis of Insertion

Let \(T\) be a (2,4) tree with \(n\) items:
- Tree \(T\) has \(O(\log n)\) height.
- Finding insertion point takes \(O(\log n)\) time because we visit \(O(\log n)\) nodes.
- Inserting the new entry takes \(O(1)\) time.
- Dealing with overflow takes \(O(\log n)\) time because each split takes \(O(1)\) time and we perform \(O(\log n)\) splits in the worst case.
- Thus, an insertion in a (2,4) tree takes \(O(\log n)\) time.

Exercise

Starting with an empty (2,4) tree, insert items with keys 1,2,3,4,5,6,7.
Exercise

inserting 4 causes a split:

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

Deleting an entry from a node \( v \) may cause an underflow, where node \( v \) becomes a 1-node with one child and no keys

Underflow and Fusion

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 \( u \)

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 \( v \) to \( u \)
3. we move an item from \( w \) to \( u \)

After a transfer, no underflow occurs

Underflow and Transfer
Analysis of Deletion

1. Let $T$ be a (2,4) tree with $n$ items.
   - Tree $T$ has $O(\log n)$ height.
2. 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.
3. Thus, deleting an item from a (2,4) tree takes $O(\log n)$ time.

Exercise delete 3,7

- Delete 3: replace by 4
- Delete 7: underflow
- Fusion

Complexity comparison

<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>simple to implement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>requires allocating a lot of memory in advance</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>
<tr>
<td></td>
<td>dynamic memory use</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>