Outline and Reading

- Multi-way search tree
  - Definition
  - Search
- (2,4) tree
  - Definition
  - Search
  - Insertion
  - Deletion
- Comparison of dictionary implementations
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 \( k_1, k_2, \ldots, k_{d-1} \)
  - 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

![Diagram of a multi-way search tree]
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 $k_1, k_2, \ldots, k_{d-1}$
  - $k = k_i$ ($i = 1, \ldots, d-1$): 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

```
          11  24
         /   \
        2    15  27  32
       /     \
      2  6  8  15  27  32
```

Example: search 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
  - $k < k_1$: 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

```
   11  24
  /   \
2  6  8  15
      /   \
     27 32
```

Example: search for 30
Multi-Way Searching

- Similar to search in a binary search tree
- At 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
  - $k < k_1$: 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
(2,4) Tree

- 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

- As opposed to a binary tree, a (2,4) tree has internal nodes with 2, 3, and 4 children.
- What is the height of the tree of n items?
- What is the big-Oh of the height of the tree of n items?
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, $h \leq \log (n + 1)$
(2,4) Tree Operations

- **Search**
  - Depends on height of tree, thus searching in a (2,4) tree with \( n \) items takes \( O(\log n) \)

- **Insert**
  - Coming up next...

- **Delete**
  - Coming up next next...
**Insertion**

- How do you insert an item into an existing tree? Ideas?

Recall the (2,4) tree properties:
- **Node-Size Property**: every internal node has at most four children
- **Depth Property**: all the external nodes have the same depth
  - THIS IS CRUCIAL TO KEEP \(O(\log N)\) SEARCH TIME - WHY?

How do you maintain these properties? Ideas?
Insertion

Let’s start at the beginning
- Insert 27
Insertion

Let’s start at the beginning

- Insert 27
- Insert 35
Insertion

Let’s start at the beginning

- Insert 27
- Insert 35
- Insert 32
- Insert 30?
Insertion

Let’s start at the beginning

- Add 30 to the node?
  - Makes 5 children = overflow…

Node-size property is broken
Insertion

Let’s start at the beginning

- We make 30 a child of the node (27, 32, 35)?
  - External node at different depths...

Depth property is broken!
Another example: insert 30 into a larger tree

We insert the new item \((k=30, o)\) at the parent \(v\) of the leaf reached by searching for \(k\)

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

What can we do?
Handling Overflows

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 $k_1$ and $k_2$ 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 $u$
## Analysis of Insertion

**Algorithm** `insertItem(k, o)`

1. We search for key `k` to locate the insertion node `v`
2. We add the new item `(k, o)` at node `v`
3. while `overflow(v)`
   - if `isRoot(v)`
     - create a new empty root above `v`
     - `v ← split(v)`

---

**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
Deletion

- How do you delete an item?
- What problems can occur?
Deletion

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

Example tree:
```
        10
       /  \
      15   24
     /    /  \
    2     12  18
   /  \
  8   12  18
```

After deletion:
```
        10
       /  \
      15   27
     /    /  \n    2     12  18
   /  \
  8   12  18
```

Example tree:
```
        27
       /  \  
      32   35
     /     /\  
    2     10  15
   /  \      /  \ 
  8   12     18   24
```

After deletion:
```
        27
       /  \  
      32   35
     /     /\  
    2     10  15
   /  \      /  \ 
  8   12     18   24
```
Deletion

What happens if I delete 12? 18?
Simply removing the node will break the depth property…
Underflow, Fusion, and Transfer

- Deleting an item 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 in the next slides.
Underflow and Fusion

Case 1: the adjacent siblings of $v$ are 2-nodes

- Fusion operation: since there is "space" in the siblings, we merge $v$ with an adjacent sibling $w$ and move an item from $u$ to the merged node $v'$
- After a fusion, the underflow may propagate to the parent $u$
**Underflow and Transfer**

- **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 item
- 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
## Summary

### Comparison of data structures and algorithms

<table>
<thead>
<tr>
<th>Data Structure</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>Skip List</td>
<td>log (n) high prob.</td>
<td>log (n) high prob.</td>
<td>log (n) high prob.</td>
<td>randomized insertion, simple 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>