Node types:
- 2-Node
  1 key
  2 children
  $a < b < c$
- 3-Node
  2 keys
  3 children
  $a < b < c < d < e$

Recap:
- AVL: Height balanced
  Binary
- 2-3 tree: Height exact
  Variable width

Identical heights
same depth

Def: A 2-3 tree of height $h$ is either:
- Empty ($h = -1$) null
- A 2-Node root and two subtrees, each 2-3 tree of height $h-1$
- A 3-Node root and three subtrees... height $h-1$

Example:
2-3 tree of height 2

Thm: A 2-3 tree of $n$ nodes has height $O(\log n)$

Roughly: $\log_3 n \leq h \leq \log_2 n$

How to maintain balance?
- Split
- Merge
- Adoption (Key rotation)

Adoption (Key Rotation):
$1 + 3 = 2 + 2$

Merge: $1 + 2/2 + 1 \rightarrow 3$

Split: $4 \rightarrow 2 + 2$

Conceptual tool:
We'll allow 1-nodes
+ 4-nodes temporary
1-node
$bd + e$ 4-node $abcdefg$
**Insertion example:**

```
Insertion example:
```

**Dictionary operations:**

- **Find** - straightforward
- **Insert** - find leaf node where key “belongs” + add it (may split)
- **Delete** - find/replacement/merge or adopt

**Implementation?**

```java
class TwoThreeNode {
    int nChildren;
    TwoThreeNode children[3];
    Key key[2];
}
```

**Find**: straight forward

```
1. Find key
2. Return
```

**Insert**: find leaf node

```
1. Insert key
2. Traverse to parent
3. Add key
```

**Delete**: find /replacement/merge or adopt

```
1. Delete key
2. Traverse to parent
3. Delete key
```

**Example (continued)**

```
Example (continued)
```

**Deletion remedy:**

- Have a 3-node neighboring sibling → adopt
- O.w.: Merge with either sibling + steal key from parent

**Example (continued)**

```
Example (continued)
```

**Adoption (rotation) preferred**
Further thoughts:
- Time? $O(\log n)$
- Can you go wider? yes! $2-3$
- Linked vs. array allocation?

![Diagram of a B-tree with keys and children](image)

- $n_{\text{child}}: 2,3$
- Keys: $x, y$
- Child: $z$