2-3-4 Tree

B- Tree
Outline

- Balanced Search Trees
  - 2-3 Trees
  - 2-3-4 Trees
  - B-Tree
2-3-4 Trees

- similar to 2-3 trees
- 4-nodes can have 3 items and 4 children

4-node

```
S  M  L
```

- Search keys < S
- Search keys > S and < M
- Search keys > M and < L
- Search keys > L
2-3-4 Tree Example
2-3-4 Tree: Insertion

Insertion procedure:
- similar to insertion in 2-3 trees
- items are inserted at the leafs
- since a 4-node cannot take another item, 4-nodes are split up during insertion process

Strategy
- on the way from the root down to the leaf: split up all 4-nodes "on the way"
  → insertion can be done in one pass (remember: in 2-3 trees, a reverse pass might be necessary)
2-3-4 Tree: Insertion

Inserting 60, 30, 10, 20, 50, 40, 70, 80, 15, 90, 100
2-3-4 Tree: Insertion

Inserting 60, 30, 10, 20 ...
2-3-4 Tree: Insertion

Inserting 50, 40 ...

... 70, ...

CSCI 2720
Slide 8
2-3-4 Tree: Insertion

Inserting 70 ...

... 80, 15 ...

(a) 30 50
   10 20 40 60

(b) 30 50
   10 20 40 60 70
2-3-4 Tree: Insertion

Inserting 80, 15 ...
2-3-4 Tree: Insertion

Inserting 90 ...

(a)  

\[
\begin{array}{c}
30 & 50 & 70 \\
10 & 15 & 20 & 40 & 60 & 80 \\
\end{array}
\]

(b)  

\[
\begin{array}{c}
30 & 50 & 70 \\
10 & 15 & 20 & 40 & 60 & 80 & 90 \\
\end{array}
\]

... 100 ...
2-3-4 Tree: Insertion

Inserting 100 ...

(a)  
50  
30  
10  15  20  40  60  80  90  

(b)  
50  
30  
10  15  20  40  60  80  90  100
2-3-4 Tree: Insertion Procedure

Splitting 4-nodes during Insertion

![Diagram of 2-3-4 Tree Insertion]

CSCI 2720
Slide 13
2-3-4 Tree: Insertion Procedure

Splitting a 4-node whose parent is a 2-node during insertion

(a)

\[
\begin{array}{c}
\text{P} \\
\text{S} \\
\text{M} \\
\text{L} \\
a \\
b \\
c \\
d \\
e \\
\end{array}
\quad \Rightarrow \quad
\begin{array}{c}
\text{M} \\
\text{P} \\
\text{S} \\
\text{L} \\
a \\
b \\
c \\
d \\
e \\
\end{array}
\]

(b)

\[
\begin{array}{c}
\text{P} \\
\text{S} \\
\text{M} \\
\text{L} \\
a \\
b \\
c \\
d \\
e \\
\end{array}
\quad \Rightarrow \quad
\begin{array}{c}
\text{P} \\
\text{M} \\
\text{S} \\
\text{L} \\
a \\
b \\
c \\
d \\
e \\
\end{array}
\]
2-3-4 Tree: Insertion Procedure

Splitting a 4-node whose parent is a 3-node during insertion

(a)

(b)

(c)
2-3-4 Tree: Deletion

Deletion procedure:
- similar to deletion in 2-3 trees
- items are deleted at the leafs
  → swap item of internal node with inorder successor
- note: a 2-node leaf creates a problem

Strategy (different strategies possible)
- on the way from the root down to the leaf:
  turn 2-nodes (except root) into 3-nodes
→ deletion can be done in one pass
  (remember: in 2-3 trees, a reverse pass might be necessary)
2-3-4 Tree: Deletion

Turning a 2-node into a 3-node ...

Case 1: an adjacent sibling has 2 or 3 items
  → "steal" item from sibling by rotating items and moving subtree
2-3-4 Tree: Deletion

Turning a 2-node into a 3-node ...

Case 2: each adjacent sibling has only one item
- "steal" item from parent and merge node with sibling
  (note: parent has at least two items, unless it is the root)

![Diagram of 2-3-4 Tree deletion process]
2-3-4 Tree: Deletion Practice

Delete 32, 35, 40, 38, 39, 37, 60
B-Trees

- Large degree B-trees used to represent very large dictionaries that reside on disk.
- Smaller degree B-trees used for internal-memory dictionaries to overcome cache-miss penalties.
AVL Trees

- \( n = 2^{30} = 10^9 \) (approx).
- \( 30 \leq \text{height} \leq 43 \).
- When the AVL tree resides on a disk, up to 43 disk access are made for a search.
- This takes up to (approx) 4 seconds.
- Not acceptable.
Red-Black Trees

- \( n = 2^{30} = 10^9 \) (approx).
- \( 30 \leq \text{height} \leq 60 \).
- When the red-black tree resides on a disk, up to 60 disk access are made for a search.
- This takes up to (approx) 6 seconds.
- Not acceptable.
m-way Search Trees

- Each node has up to $m - 1$ pairs and $m$ children.
- $m = 2 \Rightarrow$ binary search tree.
4-Way Search Tree

10
  < 30

10 < k
< 30

30 < k
< 35

k > 35

CSCI 2720
Slide 24
Maximum # Of Pairs

- Happens when all internal nodes are \( m \)-nodes.
- Full degree \( m \) tree.
- \# of nodes \( = 1 + m + m^2 + m^3 + \ldots + m^{h-1} \)
  \( = (m^h - 1)/(m - 1). \)
- Each node has \( m - 1 \) pairs.
- So, \# of pairs \( = m^h - 1 \).
### Capacity Of m-Way Search Tree

<table>
<thead>
<tr>
<th></th>
<th>m = 2</th>
<th>m = 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>h = 3</td>
<td>7</td>
<td>8 * 10^6 - 1</td>
</tr>
<tr>
<td>h = 5</td>
<td>31</td>
<td>3.2 * 10^{11} - 1</td>
</tr>
<tr>
<td>h = 7</td>
<td>127</td>
<td>1.28 * 10^{16} - 1</td>
</tr>
</tbody>
</table>
Definition Of B-Tree

- Definition assumes external nodes (extended $m$-way search tree).

- B-tree of order $m$.
  - $m$-way search tree.
  - Not empty $\Rightarrow$ root has at least 2 children.
  - Remaining internal nodes (if any) have at least $\lceil m/2 \rceil$ children.
  - External (or failure) nodes on same level.
2-3 And 2-3-4 Trees

• B-tree of order $m$.
  – $m$-way search tree.
  – Not empty $\Rightarrow$ root has at least 2 children.
  – Remaining internal nodes (if any) have at least $\lceil m/2 \rceil$ children.
  – External (or failure) nodes on same level.

• 2-3 tree is B-tree of order 3.

• 2-3-4 tree is B-tree of order 4.
B-Trees Of Order 5 And 2

• B-tree of order \( m \).
  – \( m \)-way search tree.
  – Not empty => root has at least 2 children.
  – Remaining internal nodes (if any) have at least \( \text{ceil}(m/2) \) children.
  – External (or failure) nodes on same level.

• B-tree of order 5 is 3-4-5 tree (root may be 2-node though).

• B-tree of order 2 is full binary tree.
Minimum # Of Pairs

• $n = \# \text{ of pairs.}$
• # of external nodes $= n + 1$.
• Height $= h \Rightarrow$ external nodes on level $h + 1$.

<table>
<thead>
<tr>
<th>level</th>
<th># of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$\geq 2$</td>
</tr>
<tr>
<td>3</td>
<td>$\geq$</td>
</tr>
<tr>
<td>$h + 1$</td>
<td>$2 \times \text{ceil}(m/2)$</td>
</tr>
</tbody>
</table>

$n + 1 \geq 2 \times \text{ceil}(m/2)^{h-1} \quad h \geq 1$
**Minimum # Of Pairs**

\[ n + 1 \geq 2^{\text{ceil}(m/2)^{h-1}}, h \geq 1 \]

- \( m = 200. \)

<table>
<thead>
<tr>
<th>( h )</th>
<th>( \text{# of pairs} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( \geq 199 )</td>
</tr>
<tr>
<td>3</td>
<td>( \geq 19,999 \times 10^6 - 1 )</td>
</tr>
<tr>
<td>4</td>
<td>( \geq 2 \times 10^8 - 1 )</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

\[ h \leq \log \left[ \text{ceil}(m/2) \right] \left[ \lfloor (n+1)/2 \rfloor + 1 \right] \]
Choice Of m

- Worst-case search time.
  - \((\text{time to fetch a node} + \text{time to search node}) \times \text{height}\)
  - \((a + b \cdot m + c \cdot \log_2 m) \times h\)

  where \(a, b\) and \(c\) are constants.
Insertion into a full leaf triggers bottom-up node splitting pass.
Split An Overfull Node

\[ m \ a_0 \ p_1 \ a_1 \ p_2 \ a_2 \ldots \]

- \( a_i \) is a pointer to a subtree.
- \( p_i \) is a dictionary pair.

\[ p_m \ a_m \]

\[ \text{ceil}(m/2)-1 \ a_0 \ p_1 \ a_1 \ p_2 \ a_2 \ldots \]

\[ p_{\text{ceil}(m/2)-1} \ a_{\text{ceil}(m/2)-1} \]

\[ m-\text{ceil}(m/2) \ a_{\text{ceil}(m/2)} \ p_{\text{ceil}(m/2)+1} \]

- \( a_{\text{ceil}(m/2)+1} \ldots p_m \ a_m \) plus pointer to new node
- is inserted in parent.
• Insert a pair with key = 2.
• New pair goes into a 3-node.
Insert Into A Leaf 3-node

• Insert new pair so that the 3 keys are in ascending order.

• Split overflowed node around middle key.

• Insert middle key and pointer to new node into parent.
• Insert a pair with key = 2.
• Insert a pair with key = 2 plus a pointer into parent.
• Now, insert a pair with key $= 18$. 
Insert Into A Leaf 3-node

- Insert new pair so that the 3 keys are in ascending order.

- Split the overflowed node.

- Insert middle key and pointer to new node into parent.
• Insert a pair with key = 18.
• Insert a pair with key = 17 plus a pointer into parent.
• Insert a pair with key = 17 plus a pointer into parent.
• Now, insert a pair with key = 7.
• Insert a pair with key = 6 plus a pointer into parent.
• Insert a pair with key = 4 plus a pointer into parent.
• Insert a pair with key = 8 plus a pointer into parent.

• There is no parent. So, create a new root.
• Height increases by 1.