Multi-Way Search Tree

A multi-way search tree is an ordered tree such that
- Each internal node has at least two and at most \( d \) children and stores \( d - 1 \) data items \((k_i, D_i)\)

Rule: Number of children = 1 + number of data items in a node

\[
\begin{align*}
    k_1 & \quad D_1 \\
    k_2 & \quad D_2 \\
    \ldots & \\
    k_{d-1} & \quad D_{d-1} \\
\end{align*}
\]

child 1 \hspace{1cm} child 2 \hspace{1cm} child 3 \hspace{1cm} child \( d \)

\( d \) is the degree or order of the tree
Multi-Way Search Tree

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

- Each internal node has at least two and at most $d$ children and stores $d-1$ data items $(k_i, D_i)$
- An internal node storing keys $k_1 \leq k_2 \leq \ldots \leq k_{d-1}$ has $d$ children $v_1, v_2, \ldots, v_d$ such that:

$$
\begin{align*}
&k_1 < k_1 < k_2 < k_3 < k_{d-2} < k_{d-1} < k_1 \\
&> k_1 > k_2 > k_3 > k_{d-2} > k_{d-1} > k_1
\end{align*}
$$

$$(k_i, D_i)$$
Multi-Way Search Tree

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

- Each internal node has at least two and at most \( d \) children and stores \( d - 1 \) data items \((k_i, D_i)\).
- An internal node storing keys \( k_1 \leq k_2 \leq \ldots \leq k_{d-1} \) has \( d \) children \( v_1, v_2, \ldots, v_d \) such that
- By convenience we add sentinel keys \( k_0 = -\infty \) and \( k_d = \infty \)

\[
\begin{align*}
k_0 &= -\infty \\
&\leq k_1 \\
&\leq k_2 \\
&\leq \ldots \leq k_{d-1} \\
&\leq k_d = \infty
\end{align*}
\]
Multi-Way Search Tree

A multi-way search tree is an ordered tree such that
- Each internal node has at least two and at most $d$ children and stores $d-1$ data items $(k_i, D_i)$
- An internal node storing keys $k_1 \leq k_2 \leq \ldots \leq k_{d-1}$ has $d$ children $v_1, v_2, \ldots, v_d$ such that
- By convenience we add sentinel keys $k_0 = -\infty$ and $k_d = \infty$
- The leaves store no items and serve as placeholders
Multi-Way Search Tree
Multi-Way Search Tree

Not a multiway search tree
Multi-Way Inorder Traversal

- We can extend the notion of inorder traversal from binary trees to multi-way search trees
- An inorder traversal of a multi-way search tree visits the keys in increasing order

Inorder traversal: 2, 6, 8, 11, 15, 24, 27, 30, 32
Data Structures for Multi-Way Search Trees

\[ k_0 = -\infty \leq k_1 \leq k_2 \leq \ldots \leq k_{d-1} \leq k_d = \infty \]

child 1  child 2  child 3  child d

null  D_1  D_2  \ldots  D_{d-1}  null
-
-\infty  k_1  k_2  \ldots  k_{d-1}  \infty
ch1  ch 2  ch 3  ch  d
0  1  2  d-1

node  data  keys  children
Data Structures for Multi-Way Search Trees

\[ k_0 = -\infty \leq k_1 \leq k_2 \leq \ldots \leq k_{d-1} \leq k_d = \infty \]

- child 1
- child 2
- child 3
- child d

null | D_1 | D_2 | \ldots | D_{d-1} | null

-\infty | k_1 | k_2 | \ldots | k_{d-1} | \infty

ch1 | ch 2 | ch 3 | \ldots | ch d

0 | 1 | 2 | \ldots | d-1

Secondary data structures

null D_1 D_2 \ldots D_{d-1} null

data

-\infty k_1 k_2 \ldots k_{d-1} \infty

keys

ch1 ch 2 ch 3 \ldots ch d

children

© 2014 Goodrich, Tamassia, Goldwasser (2,4) Trees
Multi-Way Searching

- Similar to search in a binary search tree
- Example: search for 31
Multi-Way Searching

- Similar to search in a binary search tree
- Example: search for 46

```
27  41  53
20  23
32  39
47
57  59  63
31
3
7  8
```

© 2014 Goodrich, Tamassia, Goldwasser (2,4) Trees
Multi-Way Searching

Algorithm \texttt{get}(r,k)

\textbf{In:} Root \( r \) of a multiway search tree, key \( k \)

\textbf{Out:} data for key \( k \) or null if \( k \) not in tree

\textbf{if} \( r \) is a leaf \textbf{then return} null

\textbf{else} {

\hspace{1em} Use binary search to find the index \( i \) such that

\hspace{2em} \( r.\text{keys}[i] \leq k < r.\text{keys}[i+1] \)

\hspace{1em} \textbf{if} \( k = r.\text{keys}[i] \) \textbf{then return} \( r.\text{data}[i] \)

\hspace{1em} \textbf{else return} \texttt{get}(r.\text{child}[i],k)

\}

(2,4) Trees
Multi-Way Searching

Algorithm get(r, k)

In: Root r of a multiway search tree, key k
Out: data for key k or null if k not in tree

if r is a leaf then return null

else {

Use binary search to find the index i such that
r.keys[i] ≤ k < r.keys[i+1]

if k = r.keys[i] then return r.data[i]
else return get(r, r.child[i])
}

Ignoring recursive calls:
c_1 \log d + c_2 operations
Time Complexity of get Operation

\[ f(n) = c + (c_1 + c_2 \log d) \times \text{height of tree} \]

is \( O(\log d \times \text{height of tree}) \)
Smallest Operation

\[ f(n) = c \times \text{height of tree} \]

is \(O(\text{height of tree})\)
Successor Operation

27
20 23
3  31
7  8
13 17
32 39
41
47
50
57 59 63
53

smallest value in subtree

successor (27)
Successor Operation

The diagram illustrates a 2,4 tree with the successor operation applied to the node containing the key 17. The successor of 17 is the node with the key 13, as it is the smallest key greater than 17 in the subtree rooted at 17.
Successor Operation

Time complexity:
get + smallest: $O(\log d \times \text{height})$, or
get + travel up: $O(\log d \times \text{height})$
so time complexity is $O(\log d \times \text{height})$
Put Operation

Assume degree = 4

insert (51)
**Put Operation**

Time Complexity:
- Find node to store new data: $O(\log d \times \text{height})$
- Add data to node: $O(d)$
- Total time complexity: $O(d + \log d \times \text{height})$
Remove Operation

```
    27  41  53
  /    |    |
20  23  /   /   /
  3  /  /   /   /
/ / /   /   /
7  8 13  17 31
| | |   |   |   |   |   |
3  7  8  13  17 31 32 39

remove(50)
```
Remove Operation

The diagram illustrates a remove operation on a (2,4) tree. The node containing the value 50 is marked, indicating that it is the target for removal.
Remove Operation

```
<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>41</td>
<td>53</td>
</tr>
<tr>
<td>57</td>
<td>59</td>
<td>63</td>
</tr>
</tbody>
</table>
```

remove(50)
Remove Operation

remove(27)
Remove Operation

```
31  41  53
  /    \
 20  23  \
 /    \
3    7  8
 /    \
3 13  17
```

```
remove(27)
```
Remove Operation

Time complexity:
get + smallest + remove data +
delete 2 nodes:
\[ O(d + \log d \times \text{height}) \]