Multi-way Search Tree

Alternative Way to achieve bounded-depth

- Extend binary node to $d$-node if it has $d$ children.
Multi-way Search Tree

Alternative Way to achieve bounded-depth

- Extend binary node to $d$-node if it has $d$ children.
- Define **multi-way search tree** (c.f. binary search tree) as follows.
  - Each internal node is a $d$-node with $d \geq 2$. Children $v_1, v_2, \ldots, v_d$. 

Each $d$ node stores $k_1 \leq k_2 \leq \cdots \leq k_{d-1}$ keys. Lead to $d$ intervals: $[-\infty, k_1], [k_1, k_2], \ldots, [k_{d-2}, k_{d-1}], [k_{d-1}, +\infty]$. 

Let $k_0 = -\infty$, $k_d = +\infty$. Each key $k_i$ stored in the subtree rooted at $v_i$ satisfies $k_{i-1} \leq k_i \leq k_i$. 

Multi-way Search Tree

Alternative Way to achieve bounded-depth

- Extend binary node to \(d\)-node if it has \(d\) children.
- Define **multi-way search tree** (c.f. binary search tree) as follows.
  - Each internal node is a \(d\)-node with \(d \geq 2\). Children \(v_1, v_2, \ldots, v_d\).
  - Each \(d\) node stores \(k_1 \leq k_2 \leq \cdots \leq k_{d-1}\) keys. Lead to \(d\) intervals:
    \[
    [-\infty, k_1], [k_1, k_2], \ldots, [k_{d-2}, k_{d-1}], [k_{d-1}, +\infty]
    \]
Multi-way Search Tree

Alternative Way to achieve bounded-depth

- Extend binary node to $d$-node if it has $d$ children.
- Define multi-way search tree (c.f. binary search tree) as follows.
  - Each internal node is a $d$-node with $d \geq 2$. Children $v_1, v_2, \cdots, v_d$.
  - Each $d$ node stores $k_1 \leq k_2 \leq \cdots \leq k_{d-1}$ keys. Lead to $d$ intervals:
    
    $$[-\infty, k_1], [k_1, k_2], \cdots, [k_{d-2}, k_{d-1}], [k_{d-1}, +\infty]$$

- Let $k_0 = -\infty$, $k_d = +\infty$. Each key $k$ stored in the subtree rooted at $v_i$ satisfies $k_{i-1} \leq k \leq k_i$. 

Multi-way Search Tree
Search in Multi-way Search Tree

Extension to the binary search

- Instead of $\leq$, $>$ relation with the key stored in each node, there are $d$ possible relations for each $d$-node.
Search in Multi-way Search Tree

Extension to the binary search

- Instead of $\leq$, $>$ relation with the key stored in each node, there are $d$ possible relations for each $d$-node.

- Let $k_1 \leq \cdots \leq k_{d-1}$ be the keys stored at a $d$-node. Then for the search key $k$, it can lie in any of the following internal:

  $[-\infty, k_1], [k_1, k_2], \cdots, [k_{d-2}, k_{d-1}], [k_{d-1}, +\infty]$
Search in Multi-way Search Tree

Extension to the binary search

- Instead of $\leq$, $>$ relation with the key stored in each node, there are $d$ possible relations for each $d$-node.
- Let $k_1 \leq \cdots \leq k_{d-1}$ be the keys stored at a $d$-node. Then for the search key $k$, it can lie in any of the following internal:

$$[-\infty, k_1], [k_1, k_2], \cdots, [k_{d-2}, k_{d-1}], [k_{d-1}, +\infty]$$

- Identify the interval. If $k = k_i$ for some $i$, we find the key. Otherwise, go the corresponding subtree.
Multi-way Search Tree: Search 12
Multi-way Search Tree: Search 24
Properties about multi-way search tree

Theorem (3.3 on page 161)

A multi-way search tree storing $n$ items has $n + 1$ external nodes.
Properties about multi-way search tree

Theorem (3.3 on page 161)

A multi-way search tree storing \( n \) items has \( n + 1 \) external nodes.

Proof.

- Proof by induction. Let \( e(n) \) be the number external nodes of a multi-way search tree storing \( n \) items.
Properties about multi-way search tree

Theorem (3.3 on page 161)

A multi-way search tree storing $n$ items has $n + 1$ external nodes.

Proof.

Proof by induction. Let $e(n)$ be the number external nodes of a multi-way search tree storing $n$ items.

$e(0) = 1, e(1) = 2, e(2) = 3.$
Properties about multi-way search tree

Theorem (3.3 on page 161)
A multi-way search tree storing $n$ items has $n + 1$ external nodes.

Proof.

- Proof by induction. Let $e(n)$ be the number of external nodes of a multi-way search tree storing $n$ items.
- $e(0) = 1$, $e(1) = 2$, $e(2) = 3$.
- Look at the root of such a multi-way search tree storing $k$ items. Assume the root is a $d$-node.
Properties about multi-way search tree

Theorem (3.3 on page 161)
A multi-way search tree storing $n$ items has $n + 1$ external nodes.

Proof.

- Proof by induction. Let $e(n)$ be the number external nodes of a multi-way search tree storing $n$ items.
- $e(0) = 1$, $e(1) = 2$, $e(2) = 3$.
- Look at the root of such a multi-way search tree storing $k$ items. Assume the root is a $d$-node.
- By definition, this root stores $d - 1$ items and has $d$ subtrees, each storing $k_i$ items (s.t., $k_i \leq k$).
Properties about multi-way search tree

Theorem (3.3 on page 161)

A multi-way search tree storing $n$ items has $n + 1$ external nodes.

Proof.

- The total number of external nodes is thus

$$
\sum_{i=1}^{d} e(k_i) = \sum_{i=1}^{d} (k_i + 1) = d + \sum_{i=1}^{d} k_i.
$$
Properties about multi-way search tree

Theorem (3.3 on page 161)

A multi-way search tree storing $n$ items has $n + 1$ external nodes.

Proof.

- The total number of external nodes is thus

\[ \sum_{i=1}^{d} e(k_i) = \sum_{i=1}^{d} (k_i + 1) = d + \sum_{i=1}^{d} k_i. \]

- The total number of storing items is

\[ (d - 1) + \sum_{i=1}^{d} k_i = n. \]
Properties about multi-way search tree

**Theorem (3.3 on page 161)**

*A multi-way search tree storing $n$ items has $n + 1$ external nodes.*

**Proof.**

- The total number of external nodes is thus

\[
\sum_{i=1}^{d} e(k_i) = \sum_{i=1}^{d} (k_i + 1) = d + \sum_{i=1}^{d} k_i.
\]

- The total number of storing items is

\[
(d - 1) + \sum_{i=1}^{d} k_i = n.
\]

- Thus, we have $e(n) = n + 1$.  

Implementation and Complexity

Primary & Secondary Data Structure

- On the high level, we use a tree structure and the search takes $O(h)$. 
- At each node, we need to perform binary search to locate the interval. Multiple choices for this one. For a sorted vector, binary search takes $O(\log d)$ for a $d$-node.
- Assume all nodes are at most $d$-nodes, then the complexity is $O(h \log d)$. 
Implementation and Complexity

Primary & Secondary Data Structure

▷ On the high level, we use a tree structure and the search takes $O(h)$.

▷ At each node, we need to perform binary search to locate the interval. Multiple choices for this one. For a sorted vector, binary search takes $O(\log d)$ for a $d$-node.
Implementation and Complexity

Primary & Secondary Data Structure

- On the high level, we use a tree structure and the search takes $O(h)$.
- At each node, we need to perform binary search to locate the interval. Multiple choices for this one. For a sorted vector, binary search takes $O(\log d)$ for a $d$-node.
- Assume all nodes are at most $d$-nodes, then the complexity is $O(h \log d)$. 
Performance depends on $h$ and $d$

For efficient multi-way tree implementation, we need small $h$ and $d$. What is the best trade-off?
Performance depends on $h$ and $d$

For efficient multi-way tree implementation, we need small $h$ and $d$. What is the best trade-off?

(2,4) Trees

- Achieve $h = \Theta(\log(n))$ and $2 \leq d \leq 4$. 
Performance depends on $h$ and $d$

For efficient multi-way tree implementation, we need small $h$ and $d$. What is the best trade-off?

(2,4) Trees

- Achieve $h = \Theta(\log(n))$ and $2 \leq d \leq 4$.
- **Size Property**: every node has at most four children.
- **Depth Property**: all the external nodes have the same depth.
Performance depends on $h$ and $d$

For efficient multi-way tree implementation, we need small $h$ and $d$. What is the best trade-off?

(2,4) Trees

- Achieve $h = \Theta(\log(n))$ and $2 \leq d \leq 4$.
- **Size Property**: every node has at most four children.
- **Depth Property**: all the external nodes have the same depth.
- Size and Depth Properties $\Rightarrow h = \Theta(\log(n))$. 
The height of a (2,4) tree storing \( n \) items is \( \Theta(\log(n)) \).
The height of a (2,4) tree storing \( n \) items is \( \Theta(\log(n)) \).

Proof.

- Depth Property \( \Rightarrow \) no short subtrees.
Theorem (3.4 on page 163)

The height of a (2,4) tree storing \(n\) items is \(\Theta(\log(n))\).

Proof.

- Depth Property \(\Rightarrow\) no short subtrees.
- Let \(h\) be the height. The \# of nodes at depth \(i\) is between \(2^i\) and \(4^i\).
Theorem (3.4 on page 163)

The height of a (2,4) tree storing $n$ items is $\Theta(\log(n))$.

Proof.

- Depth Property $\Rightarrow$ no short subtrees.
- Let $h$ be the height. The # of nodes at depth $i$ is between $2^i$ and $4^i$.
- Thus, the # of external nodes is between $2^h$ and $4^h$. 

$h = \Theta(\log(n))$ for (2,4) Trees
The height of a (2,4) tree storing $n$ items is $\Theta(\log(n))$.

Proof.

- Depth Property $\Rightarrow$ no short subtrees.
- Let $h$ be the height. The # of nodes at depth $i$ is between $2^i$ and $4^i$.
- Thus, the # of external nodes is between $2^h$ and $4^h$.
- By Theorem 3.3, i.e., the # of external nodes is $n + 1$, and hence,

  \[2^h \leq n + 1 \leq 4^h.\]
The height of a (2,4) tree storing \( n \) items is \( \Theta(\log(n)) \).

Proof.

- Depth Property \( \Rightarrow \) no short subtrees.
- Let \( h \) be the height. The \# of nodes at depth \( i \) is between \( 2^i \) and \( 4^i \).
- Thus, the \# of external nodes is between \( 2^h \) and \( 4^h \).
- By Theorem 3.3, i.e., the \# of external nodes is \( n + 1 \), and hence,

\[
2^h \leq n + 1 \leq 4^h.
\]

Then \( h = \Theta(\log(n)) \).