Multiway Search Trees

Douglas Wilhelm Harder, M.Math. LEL
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

ece.uwaterloo.ca
dwharder@alumni.uwaterloo.ca

Outline

In this topic we will look at:

- An introduction to multiway search trees
- An implementation in C++
- In-order traversals of multiway trees

Binary Tree

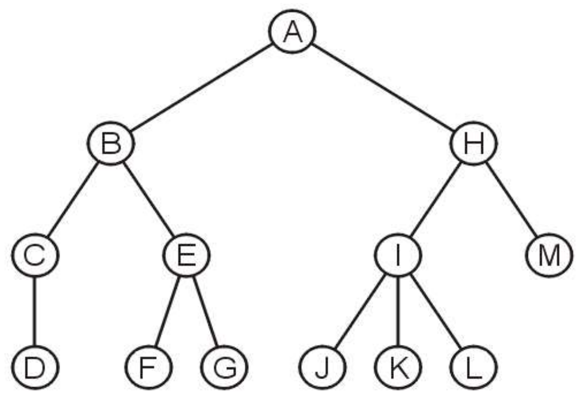
- **Full Binary Tree**
- **Complete Binary Tree**
- **Perfect Binary Tree**

https://www.geeksforgeeks.org/binary-tree-set-3-types-of-binary-tree/

6.4.1

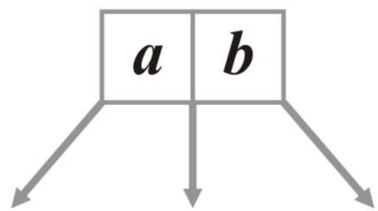
In-order traversals on general trees

We have noted that in-order traversals only make sense for binary search trees and not N-ary trees in general



3-Way Trees

Suppose we had a node storing two values and with three sub-trees:



3-Way Trees

This could be implemented as follows:

```
template <typename Type>
class Three_way_node {
    Three_way_node   *p_left_tree;
    Type              first_value;
    Three_way_node   *p_middle_tree;
    Type              second_value;
    Three_way_node   *p_right_tree;
    // ...
};
```

left_element right_element

p_left_tree p_middle_tree p_right_tree

6.4.2 3-Way Trees

In order to define a search tree, we will require that:

- The first element is less than the second element
- All sub-trees are 3-way trees
- The left sub-tree contains items less than the 1st element
- The middle sub-tree contains items between the two elements
- The right sub-tree contains items greater than the 2nd element

`left_element` `right_element`

`p_left_tree` `p_middle_tree` `p_right_tree`

3-Way Trees

If a node has only one element, all trees are assumed to be empty

- If a second object is inserted, it will be inserted into this node

```
template <typename Type>
class Three_way_node {
    Three_way_node   *p_left_tree;
    Type              first_value;
    Three_way_node   *p_middle_tree;
    Type              second_value;
    Three_way_node   *p_right_tree;

    int num_values;   # 1 or 2
    // ...
};

template <typename Type>
bool Three_way_node::full() const {
    return num_values == 2;
}
```

6.4.2.1

3-Way Trees

Most operations are more complex than with binary trees…

```
template <typename Type>
Three_way_node *Three_way_node<Type>::find( Type const &obj ) const {
    if ( !full() ) {
        return ( first() == obj );
    }

    if ( (obj == first()) || (obj == second()) ) {
        return this;
    } else if ( obj < first() ) {
        return (  left() == nullptr) ? nulltpr :   left()->find( obj );
    } else if ( obj > second()) ) {
        return ( right() == nullptr) ? nullptr :  right()->find( obj );
    } else {
        return (middle() == nulltpr) ? nullptr : middle()->find( obj );
    }
}
```

6.4.2.1

3-Way Trees

Insertion also becomes much more interesting

```
template <typename Type>
bool Three_way_node<Type>::insert( Type const &obj ) {
    if ( !full() ) {
        if ( obj == first() ) {
             return false;
        } else if ( obj < first() ) {
             second_value = first();
             first_value = obj;
        } else {
             second_value = obj;
        }

        num_values = 2;
        return true;
    }
```

3-Way Trees

```
    if ( obj == first() || obj == second() ) {
        return false;
    }

    if ( obj < first() ) {
        if ( left() == nullptr ) {
            p_left_tree = new Three_way_node( obj );
            return true;
        } else {
            return left()->insert( obj );
        }
    } else if ( obj > second() ) {
        // create or insert a new node at the right sub-tree
    } else {
        // create or insert a new node at the middle sub-tree
    }
}
```

Erasing an element is even more complex

– There are many more cases to consider

6.4.2.2

Insertion into 3-Way Trees

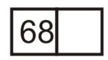
Consider inserting values into an empty 3-way tree:

- Starting with 68, it would be inserted into the root

6.4.2.2

Insertion into 3-Way Trees

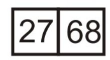
If 27 was inserted next, it would be fit into the root node



6.4.2.2

Insertion into 3-Way Trees

If 27 was inserted next, it would be fit into the root node

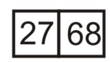


6.4.2.2

Insertion into 3-Way Trees

Any new insertion would create an appropriate sub-tree

- Inserting 91, we note that 91 > 68, so a right sub-tree is constructed



6.4.2.2

Insertion into 3-Way Trees

Any new insertion would create an appropriate sub-tree

- Inserting 91, we note that 91 > 68, so a right sub-tree is constructed

Insertion into 3-Way Trees

If we insert 38, we note that 28 < 38 < 68 and thus build a new sub-tree in the middle

6.4.2.2

Insertion into 3-Way Trees

If we insert 38, we note that 28 < 38 < 68 and thus build a new sub-tree in the middle

6.4.2.2

Insertion into 3-Way Trees

At this point, if we insert 82, we note 82 > 68 and the right sub-tree is not yet full

Insertion into 3-Way Trees

At this point, if we insert 82, we note 82 > 68 and the right sub-tree is not yet full

6.4.2.2

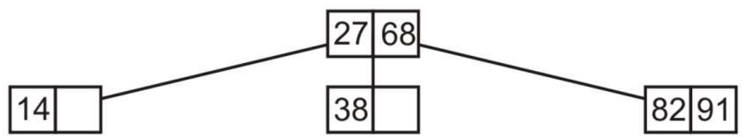
Insertion into 3-Way Trees

If we insert 14, we note 14 < 27, so we create a new node

6.4.2.2

Insertion into 3-Way Trees

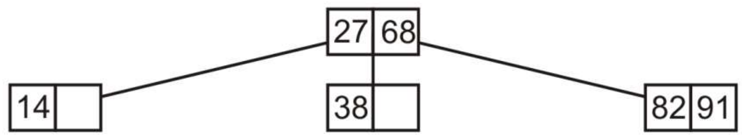
If we insert 14, we note 14 < 27, so we create a new node



6.4.2.2

Insertion into 3-Way Trees

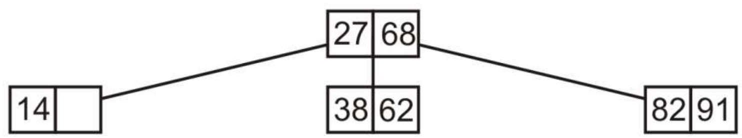
Next, inserting 62, 27 < 62 < 28 so we insert it into the middle sub-tree which also is not full



6.4.2.2

Insertion into 3-Way Trees

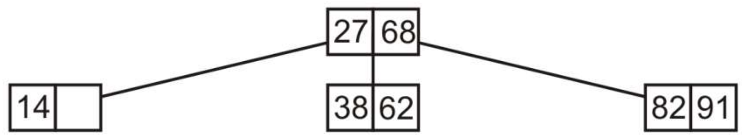
Next, inserting 62, 27 < 62 < 28 so we insert it into the middle sub-tree which also is not full



Insertion into 3-Way Trees

If we insert 45,

- First, 27 < 45 < 68 and then 38 < 45 < 62

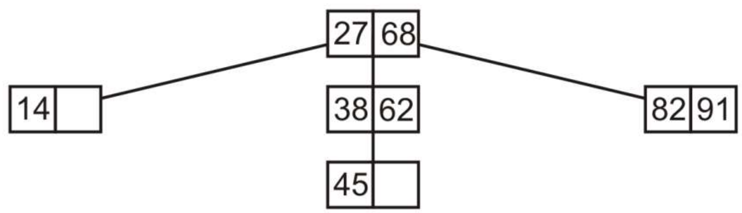


6.4.2.2

Insertion into 3-Way Trees

If we insert 45,

– First, 27 < 45 < 68 and then 38 < 45 < 62

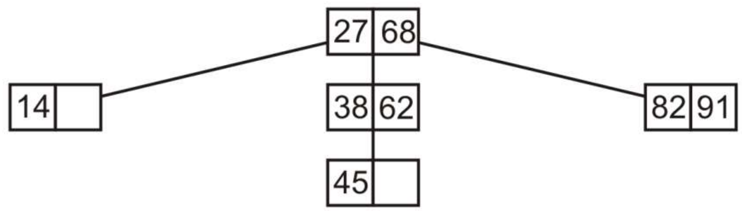


6.4.2.2

Insertion into 3-Way Trees

If we insert 76, we note 68 > 76 but then 76 < 82

– Create a new left sub-tree of the 82-91 node

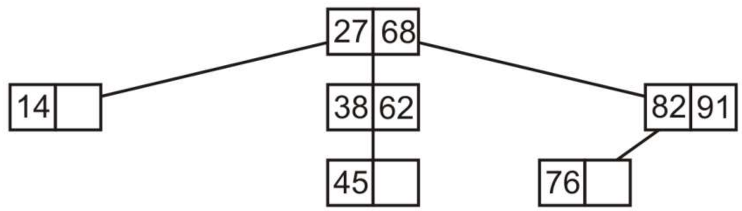


6.4.2.2

Insertion into 3-Way Trees

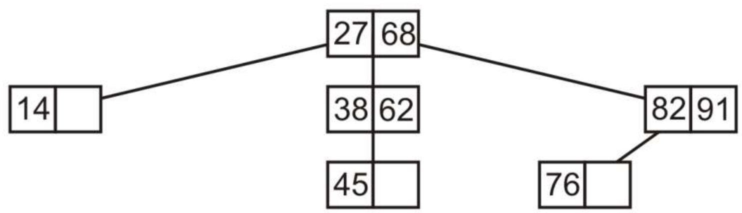
If we insert 76, we note 68 > 76 but then 76 < 82

- Create a new left sub-tree of the 82-91 node



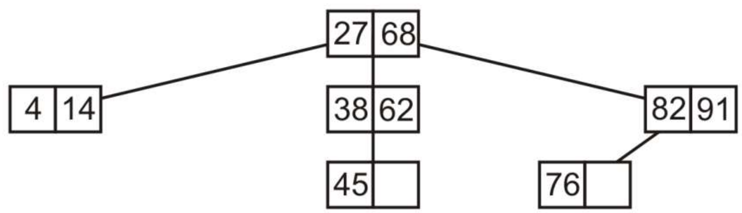
Insertion into 3-Way Trees

If we insert 4, 4 < 27 and the left sub-tree contains only a single element



Insertion into 3-Way Trees

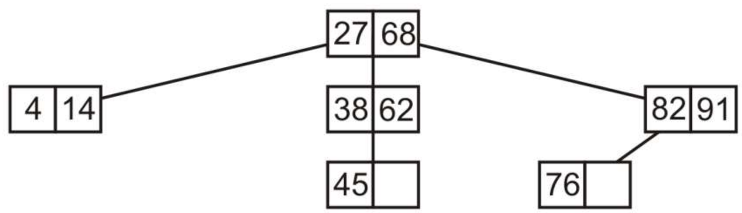
If we insert 4, 4 < 27 and the left sub-tree contains only a single element



6.4.2.2

Insertion into 3-Way Trees

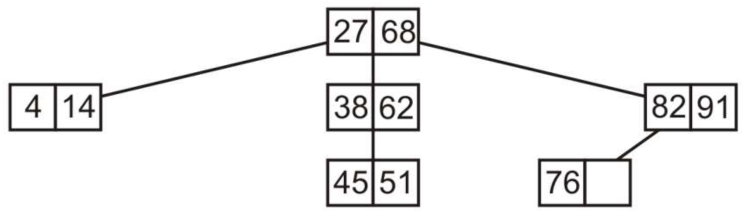
If we insert 51, 27 < 51 < 68 and 38 < 51 < 62; therefore, we insert 51 into the node containing 45



6.4.2.2

Insertion into 3-Way Trees

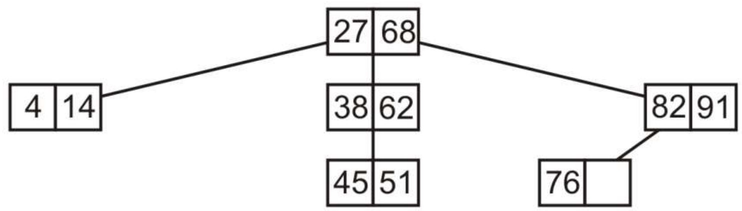
If we insert 51, 27 < 51 < 68 and 38 < 51 < 62; therefore, we insert 51 into the node containing 45



Insertion into 3-Way Trees

If we insert 8, 8 < 27 and then 4 < 8 < 14

– Construct a new middle sub-tree of the 4-14 node

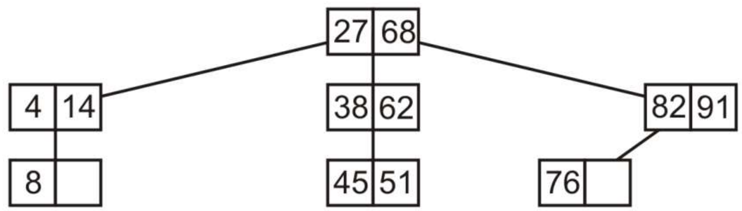


6.4.2.2

Insertion into 3-Way Trees

If we insert 8, 8 < 27 and then 4 < 8 < 14

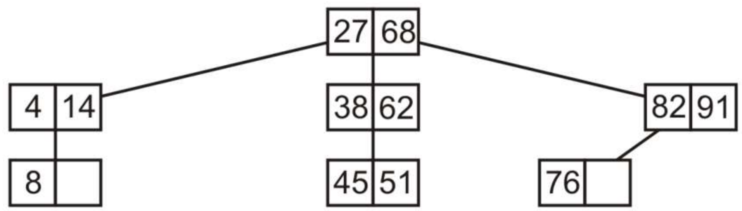
- Construct a new middle sub-tree of the 4-14 node



Insertion into 3-Way Trees

If we insert 98, 98 > 68 and 98 > 91

– Construct a new right sub-tree of the 81-91 node

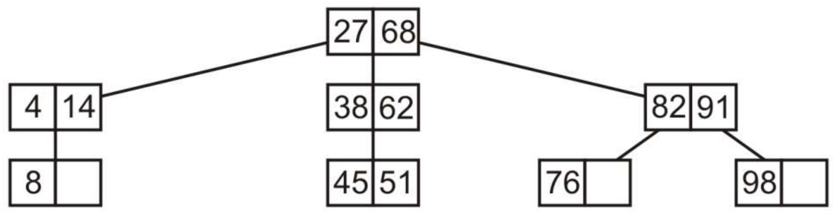


6.4.2.2

Insertion into 3-Way Trees

If we insert 98, 98 > 68 and 98 > 91

– Construct a new right sub-tree of the 81-91 node

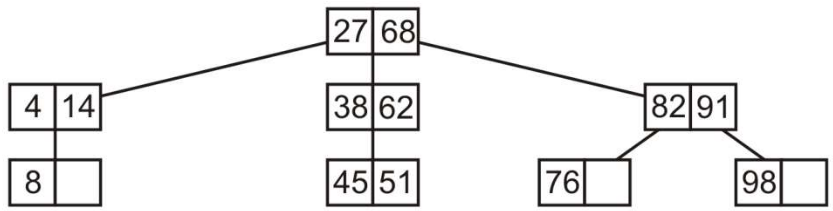


6.4.2.2

Insertion into 3-Way Trees

Finally, consider adding 57:

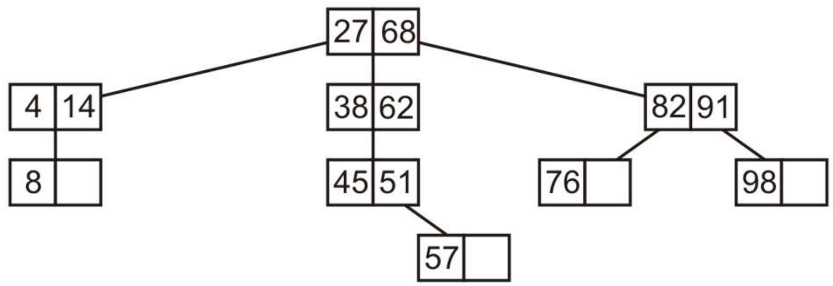
- 27 < 57 < 68, 38 < 57 < 62 and 57 > 51
- Construct a new right sub-tree of the 45-51 node



Insertion into 3-Way Trees

Finally, consider adding 57:

- 27 < 57 < 68, 38 < 57 < 62 and 57 > 51
- Construct a new right sub-tree of the 45-51 node



6.4.2.1

In-order Traversals

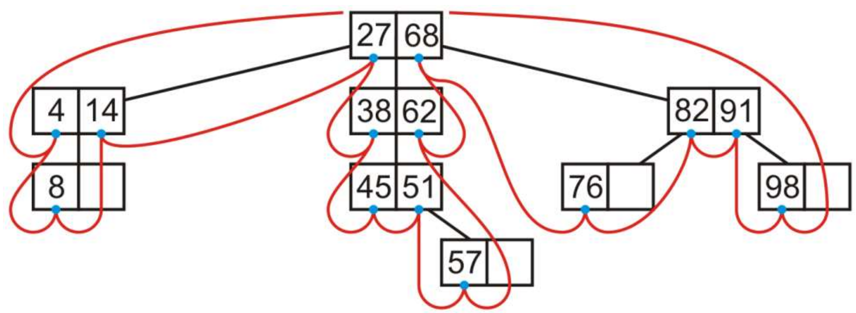
Insertion also becomes much more interesting

```
template <typename Type>
void Three_way_node<Type>::in_order_traversal() const {
    if ( !full() ) {
        cout << first();
    } else {
        if ( left() != nullptr ) {
            left()->in_order_traversal();
        }

        cout << first();

        if ( middle() != nullptr ) {
            middle()->in_order_traversal();
        }

        cout << second();

        if ( right() != nullptr ) {
            right()->in_order_traversal();
        }
    }
}
```

6.4.2.3

In-order Traversals

An in-order traversal can be performed on this tree:



4 8 14 27 38 45 51 57 62 68 76 82 91 98

Multiway tree implementation

Suppose we had a node storing $N-1$ values and with N sub-trees

- We will describe this as an N-way tree

```
template <typename Type, int N>
class Multiway_node {
    private:
        int num_values;
        Type elements[N – 1];
        Multiway_node *[N];  // an array of pointers to multiway nodes
    public:
        Multiway_node( Type const & );
        // ...
};

template<typename Type, int M>
bool M_ way_node<Type, M>::full() const {
    return ( num_values == M - 1 );
}
```

6.4.3

Multiway tree implementation

The constructor would initial the node to store one element

```
template <typename Type, int N>
Multiway_node<Type, N>::Multiway_node( Type const &obj ):
num_values( 1 ) {
    elements[0] = obj;

    // All sub-treees are null sub-trees
    for ( int i = 0; i < N; ++i ) {
        subtrees[i] = nullptr;
    }
}
```

Multiway tree implementation

An in-order traversal would be similar:

```
template <typename Type, int N>
void Multiway_node<Type, N>::in_order_traversal() const {
    if ( empty() ) {
        return;
    } else if ( !full() ) {
        for ( int i = 0; i < num_values; ++i ) {
            cout << elements[i];
        }
    } else {
        for ( int i = 0; i < N - 1; ++i ) {
            if ( subtrees[i] != nullptr ) {
                subtrees[i]->in_order_traversal();
            }
            cout << elements[i];
        }

        subtrees[N - 1]->in_order_traversal();
    }
}
```

6.4.3.1

Size

Question:

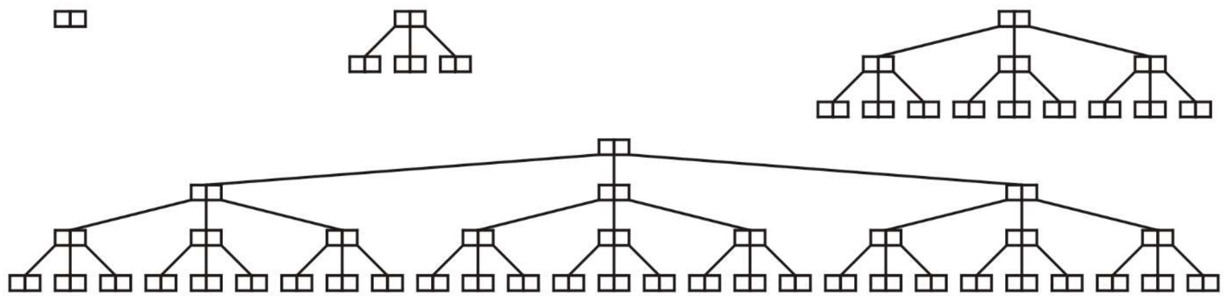
– What is the maximum number of elements which may be stored in a multiway tree of height h?

We will consider 3-way trees and, if possible, generalize

6.4.3.1

Size

Examining these perfect 3-way trees



we get the table:

h	Size
0	2
1	8
2	26
3	80

6.4.3.1

Size

Suggested form:

- The maximum number of nodes in a perfect multiway tree of height h is $N^{h+1} - 1$

Observations

- This is true when $N = 2$: $2^{h+1} - 1$

To prove this, we need only observe:

- A perfect N-ary tree of height h has $\frac{N^{h+1}-1}{N-1}$ nodes
- Thus, if each node now has $N - 1$ elements:

$$\frac{N^{h+1}-1}{N-1}(N-1) = N^{h+1} - 1$$

Size

Note also that the majority of elements are in the leaf nodes:

- There are N^h leaf nodes in a perfect M-way search tree of height h
- Each of these stores $N - 1$ elements

Thus, we may calculate the ratio

$$\frac{N^h(N-1)}{N^{h+1}-1} \approx \frac{N^h(N-1)}{N^{h+1}} = \frac{N-1}{N}$$

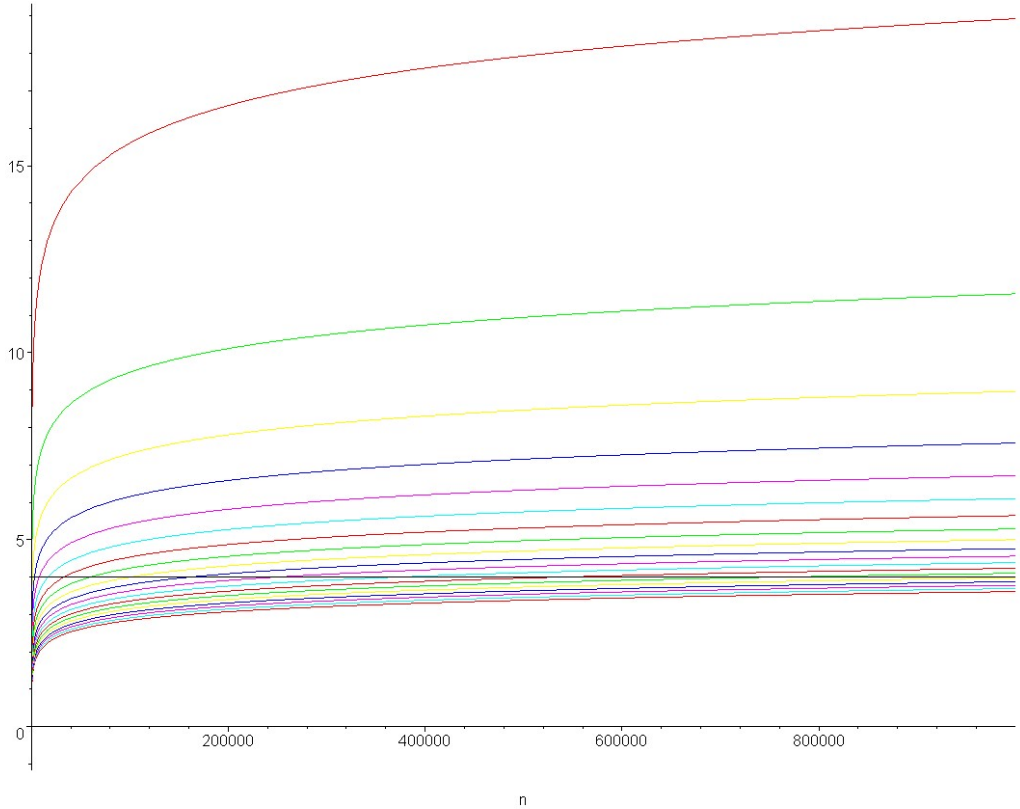
For example:

- In an 8-way search tree, ~87.5 % of elements are in leaf nodes
- In a 100-way search tree, ~99 % of elements are in the leaf nodes

Minimum height

The minimum height of a multiway tree storing n elements is $\lfloor \log_N(n) \rfloor$

- For large N, the depth is potentially much less than a binary tree
- A plot of the minimum height of a multiway tree for $N = 2, 3, ..., 20$ for up to one-million elements

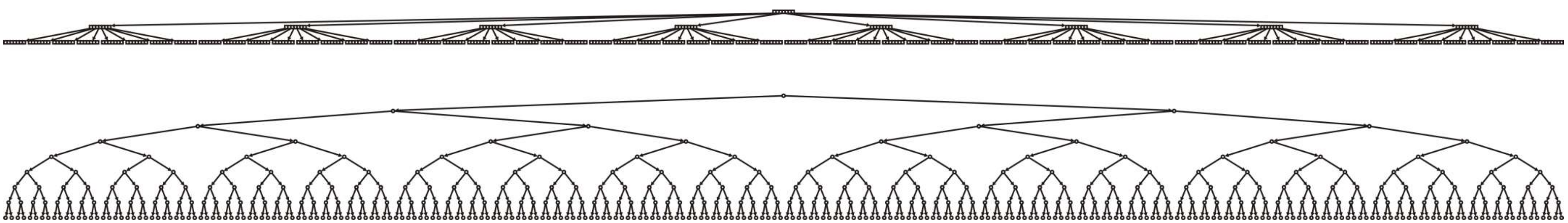


6.4.3.3

8-way trees versus binary trees

Compare:

- A perfect 8-way tree with $h = 2$
  - 511 elements in 73 nodes
- A perfect binary tree with $h = 8$
  - 511 elements in 511 nodes

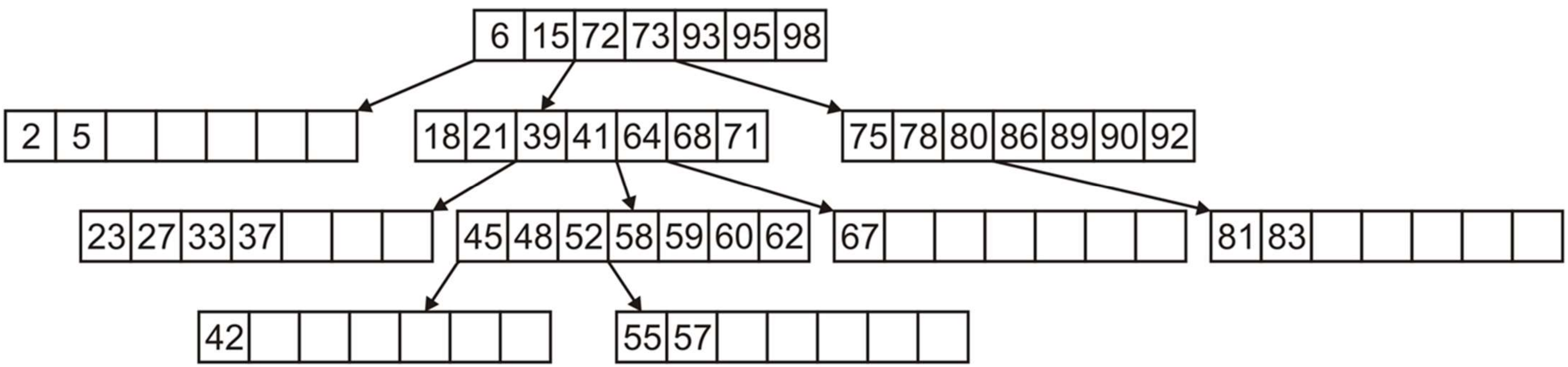


6.4.3.4

8-way tree example

A sample 8-way search tree:

- Note how a binary search is required to find the appropriate sub-tree
- How do you determine if 43 is in this search tree?
- Question: what order would these entries have been inserted?
- How do we erase an element?



6.4.3.4

Multiway trees

Advantage:

- Shorter paths from the root

Disadvantage:

- More complex

Under what conditions is the additional complexity worth the effort?

- When the cost from jumping nodes is exceptionally dominant

Summary

In this topic, we have looked at:

- Multiway trees
  - Each node stores $N - 1$ sorted elements
  - N sub-trees interleave the elements
  - Perfect Multiway trees store $N^{h+1} - 1$ elements
- We saw an implementation in C++
- We considered in-order traversals of multiway trees
- Has the potential to store more elements in shallower trees

References

[1] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, MIT Press, 1990, §7.1-3, p.152.

[2] Weiss, *Data Structures and Algorithm Analysis in C++*, *3rd Ed.*, Addison Wesley, §6.5-6, p.215-25.

Usage Notes

- These slides are made publicly available on the web for anyone to use
- If you choose to use them, or a part thereof, for a course at another institution, I ask only three things:
  - that you inform me that you are using the slides,
  - that you acknowledge my work, and
  - that you alert me of any mistakes which I made or changes which you make, and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides

Sincerely,

Douglas Wilhelm Harder, MMath

dwharder@alumni.uwaterloo.ca