M-Way Trees

- In this topic we will look at:
  - In-order traversals of binary trees
  - Limitations of in-order traversals with *n*-ary trees
  - Introduction to *M*-way trees
  - In-order traversals of *M*-way trees

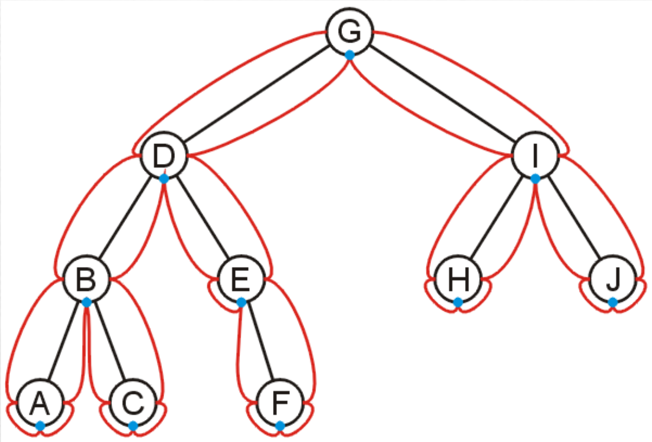
الدكتور
اثير العاني

In-order Traversals

- Two depth-first traversals:
  - Pre-order
  - Post-order
- First and last visits during an Euler walk

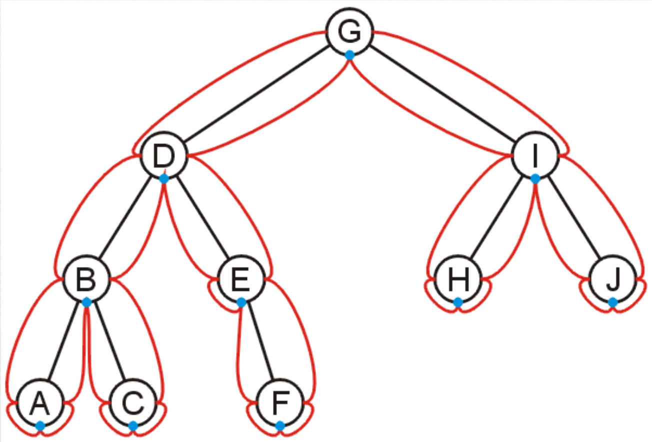
In-order Traversals

- For binary trees, there is a third intermediate visit
  - An *in-order* depth-first traversal



In-order Traversals

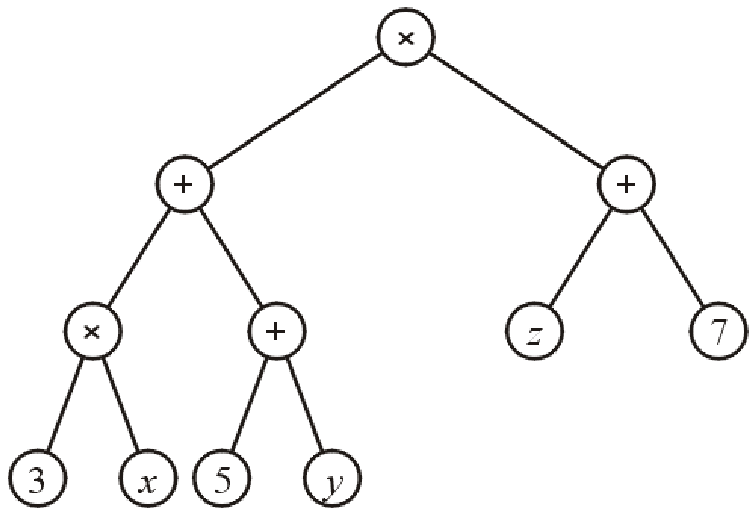
- This visits a binary search tree in order



A, B, C, D, E, F, G, H, I, J

In-order Traversals

- Printing an expression tree using in-fix notation



$(3x + 5 + y)(z + 7)$

Application

```
class Algebraic;
void pretty_print( Algebraic * parent ) {
    if ( !leaf() ) {
        // If we are printing an operator (not a leaf node) then
        // we want to print an opening parenthesis if the parent
        // operator has higher precedence, e.g.,
        //       *                              +
        //    +      y   ->  (x + 5) y       *       y   ->  5x + y
        // x   5                          5   x

        if ( parent->precedence() > precedence() ) {
            cout << "(";
        }  // pre-order visit

        left_tree->pretty_print( this ); // traverse left tree
    }
```

Application

```
// If we are printing a multiplication, then we will
// print a star iff both sub-trees are numeric values, e.g.
//      *                 *                       *
//    3   y  ->  3y    4      +   ->  4(x + 5)   3   5 -> 3*5
//                          x   5

if ( is_multiplication() ) {
    if ( left_tree->numeric() && right_tree->numeric() ) {
        cout << "*";
    }
} else {
    cout << this;  // print this object
}
```

Application

```
if ( !leaf() ) {
    right_tree->pretty_print( this ); // traverse right sub-tree

  // If we are printing an operator (not a leaf node) then
  // we want to print an opening parenthesis if the parent
  // operator has higher precedence, e.g.,
  //      *                                 +
  //   +       y   ->  (x + 5) y         *        y   ->  5x + y
  // x   5                            5   x

    if ( parent->precedence() > precedence() ) {
        cout << ")";
    }  // post-order visit
}
```

3-Way Trees

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

```
template<class Object>
class Three_way_node {
    Three_way_node   *left_tree;
    Object            first_element;
    Three_way_node   *middle_tree;
    Object            second_element;
    Three_way_node   *right_tree;
    // ...
};
```

left_element   right_element

left_tree   middle_tree   right_tree

3-Way Trees

- We will require that
  - 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

3-Way Trees

- One immediate consequence is that the first element is less than the second
- Problem: we may not be able to fill both entries in a node:
  - Require that the right sub-tree is empty if the node contains only one element (the first)

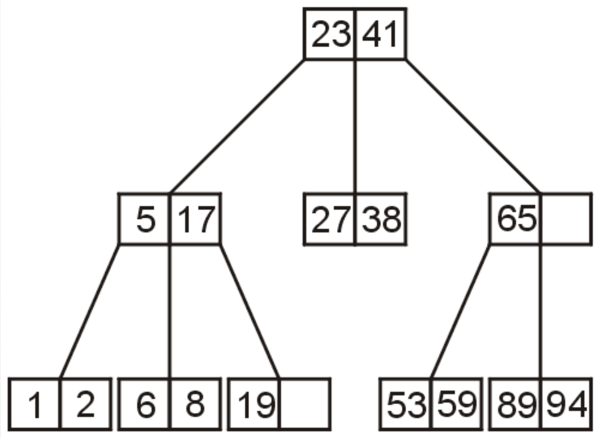
3-Way Trees

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

```
template<class Object>
class Three_way_node {
    Three_way_node   *left_tree;
    Object            first_element;
    Three_way_node   *middle_tree;
    Object            second_element;
    Three_way_node   *right_tree;
    int num_elements;   # 1 or 2
    // ...
};
```

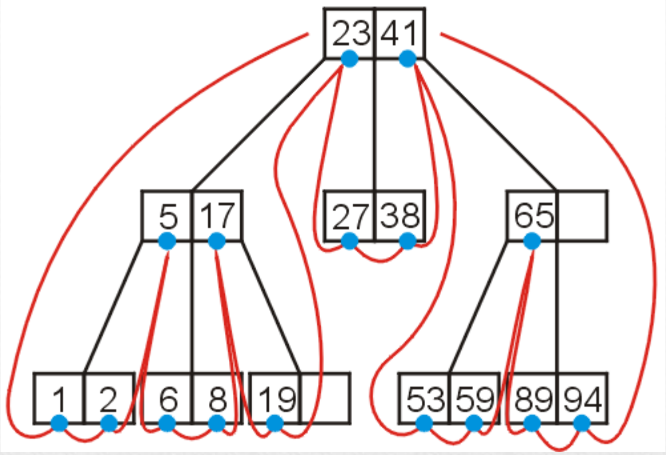
3-Way Trees

- An example of a 3-way tree:



3-Way Tree

- An in-order traversal now makes sense:



1 2 5 6 8 17 19 23 27 38 41 53 59 65 89 94

M-Way Trees

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

```
template<class Object>
class M_way_node {
    private:
        int M;
        int num_elements;
        Object *elements;
        M_way_node **subtrees;
        // for an array of M pointers to M-way nodes
    public:
        // ...
};
```

M-Way Trees

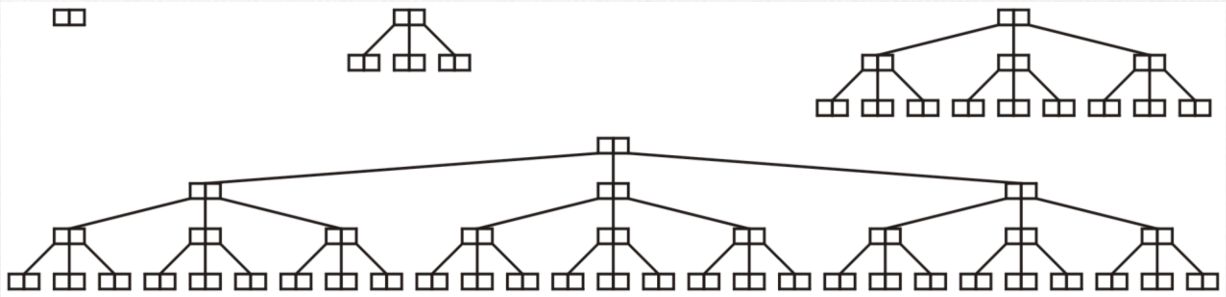
```
template<class Object>
M_way_node<Object>::M_way_node( const Object &obj, int m ):
    M( m ),
    num_elements( 1 ),
    elements( new Object[M - 1] ),
    subtrees( new M_way_node<Object> *[M] )
{
    elements[0] = obj;

    for ( int i = 0; i < M; ++i ) {
        subtrees[i] = 0;
    }
}
```

M-Way Trees

- Question:
  - What is the maximum number of elements which may be stored in an M-way tree of height h?
- Consider the 3-way trees and, if possible, generalize

M-Way Trees

- Examining these perfect 3-way trees



we get the table:

h	count	formula
0	2	$3^1 - 1$
1	8	$3^2 - 1$
2	26	$3^3 - 1$
3	80	$3^4 - 1$

M-Way Trees

- Suggested form:
  - The maximum number of nodes in a perfect M-way tree of height h is $M^{h+1} - 1$
- Observations
  - This is true when $M = 2$: $2^{h+1} - 1$
- To prove that this is true in general, we will first make use of one fact...

M-Way Trees

- We will require the following:
  - the maximum number of leaf nodes in an M-way tree of height h is M^h
- Proof (by induction):
  - when $h = 0$, there is $M^0 = 1$ node (a leaf node)
  - assume for $h = k$ that there are M^k leaf nodes
  - for $h = k + 1$, each leaf node has M children:

$$M^k M = M^{k+1}$$

Q.E.D.

M-Way Trees

- Similarly, we will show that the maximum number of elements which may be stored in M-way tree of height h is
- First, when $h = 0$, the formula is $M^1 - 1$ which is the maximum number of elements a single node can store

M-Way Trees

- We will assume the statement is true for $h = k$, that is, the maximum number of elements is $M^{k+1} - 1$
- A tree of height $h = k$ has M^k leaf nodes, and therefore, if each of these have the maximum number of children (M), we therefore have $M \cdot M^k$ leaf nodes, each of which stores $M - 1$ elements

M-Way Trees

- Therefore, the maximum number of elements stored is in a tree of height
  $h = k + 1$ is:
  - the total number of elements stored in a tree of height $h = k$ plus
  - $M - 1$ for each possible sub-tree of each leaf node of height $h = k$
- That is, $M^{k+1} - 1 + MM^k(M - 1) =$
  $M^{k+1} - 1 + M^{k+2} - M^{k+1} - 1 = M^{k+2} - 1$

M-Way Trees

- Thus, the statement must be true for all $h > 0$
- One nice consequence is that the minimum height of an M-way tree which stores n elements is given by $\lfloor \log_M(n) \rfloor$

M-Way Trees

- An M-way tree has the following properties:
  - each node has k elements where $1 \leq k < M$ and $e_0 < \cdots < e_{k-1}$
  - each node has at most one $k + 1$ sub-trees $T_0, T_1, \ldots, T_k$ such that:
    - all elements ε in the sub-tree T_0 satisfy $\varepsilon < e_0$,
    - all elements ε in T_j ($j = 1, \ldots, k - 1$) satisfy:

$$e_{j-1} < \varepsilon < e_j$$

    - all elements ε in the sub-tree T_k satisfy $\varepsilon > e_{k-1}$

M-Way Trees

- Observations
  - we require that the elements in a given node are filled in order
  - intermediate trees may be empty
  - a binary search tree is a 2-way tree
  - the minimum depth of an M-way tree with n nodes is $\log_M(n + 1) - 1$
  - potentially much less depth than a binary tree

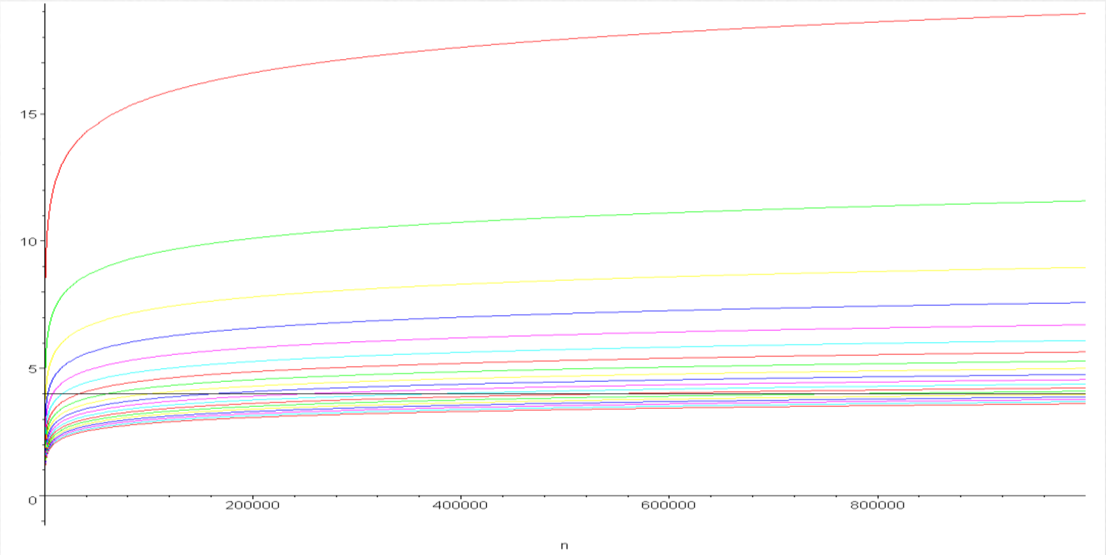
M-Way Trees

- Most keys are stored in the leaves
  - M^h leaves
  - A total of $M - 1$ keys per leaf
- Thus

$$(M-1)M^h/(M^{h+1}-1) \approx (M-1)/M$$

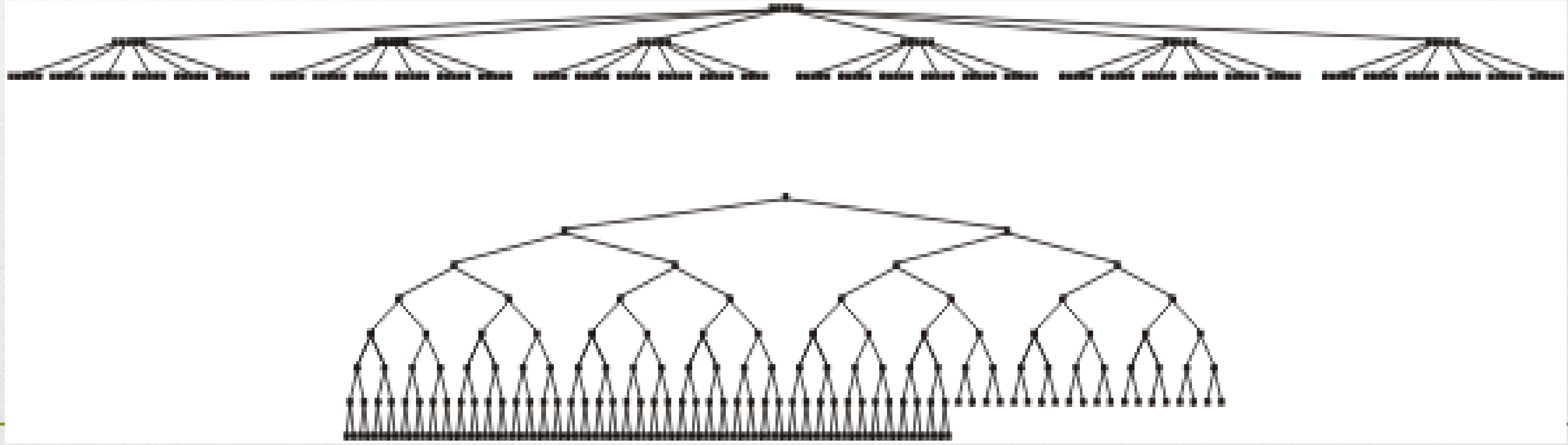
M-Way Trees

- A plot of the minimum height of an *M*-way tree for *M* = 2, 3, ..., 20 for up to one-million elements



M-Way Trees

- Compare:
  - A perfect 6-way tree with $h = 2$
    - 215 elements in 43 nodes
  - A complete binary tree with $n = 215$ and $h = 7$



M-Way 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:
  - In-order depth-first traversals
  - Limitations on n-ary trees
  - M-way trees