

Tree Traversals & Maps



Nelson Padua-Perez
Chau-Wen Tseng

Department of Computer Science
University of Maryland, College Park

Tree Traversal

■ Goal

- Visit every node in binary tree

■ Approaches

■ Depth first

- Preorder ⇒ parent before children
- Inorder ⇒ left child, parent, right child
- Postorder ⇒ children before parent

- Breadth first ⇒ closer nodes first

Tree Traversal Methods

■ Pre-order

1. Visit **node** // first
2. Recursively visit left subtree
3. Recursively visit right subtree

■ In-order

1. Recursively visit left subtree
2. Visit **node** // second
3. Recursively visit right subtree

■ Post-order

1. Recursively visit left subtree
2. Recursively visit right subtree
3. Visit **node** // last

Tree Traversal Methods

■ Breadth-first

```
BFS(Node n) {  
    Queue Q = new Queue();  
    Q.enqueue(n);           // insert node into Q  
    while ( !Q.empty() ) {  
        n = Q.dequeue();    // remove next node  
        if ( !n.isEmpty() ) {  
            visit(n);       // visit node  
            Q.enqueue(n.Left()); // insert left subtree in Q  
            Q.enqueue(n.Right()); // insert right subtree in Q  
        }  
    }  
}
```

Tree Traversal Examples

■ Pre-order (prefix)

■ $+ \times 2 3 / 8 4$

■ In-order (infix)

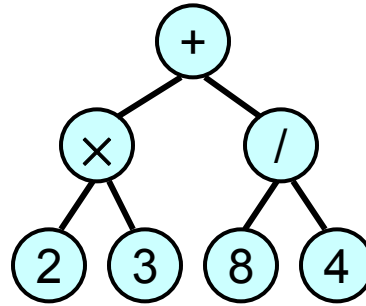
■ $2 \times 3 + 8 / 4$

■ Post-order (postfix)

■ $2 3 \times 8 4 / +$

■ Breadth-first

■ $+ \times / 2 3 8 4$



Expression tree

Tree Traversal Examples

■ Pre-order

■ 44, 17, 32, 78,
50, 48, 62, 88

■ In-order

■ 17, 32, 44, 48,
50, 62, 78, 88

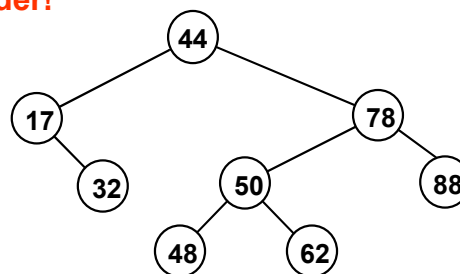
■ Post-order

■ 32, 17, 48, 62,
50, 88, 78, 44

■ Breadth-first

■ 44, 17, 78, 32,
50, 88, 48, 62

Sorted
order!

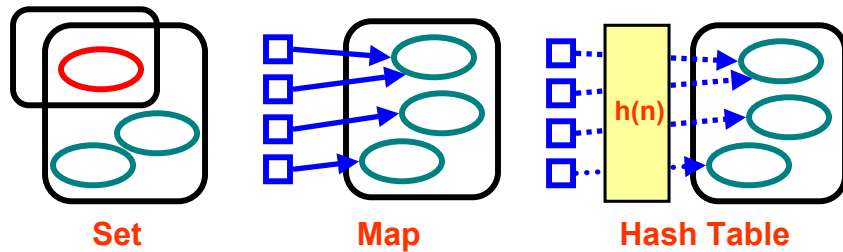


Binary search tree

Set Data Structures

Types

- Set
- Map
- Hash Table



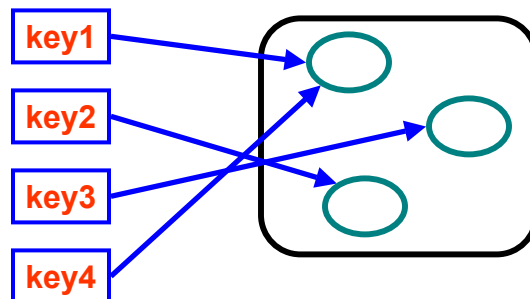
Maps

Map (associative array)

- Unordered collection of objects (values)
- One or more **keys** associated with each object
- Can use key to retrieve object

Example

- $A["key1"] = \dots$



Maps

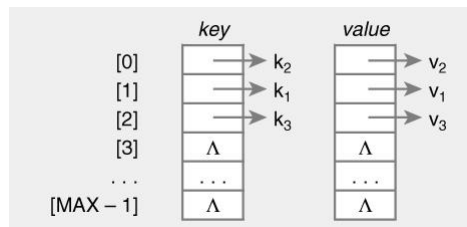
■ Methods

- **Void Put(Key, Object)** // inserts element
- **Object Get(Key)** // returns element
- **Void Remove(Key)** // removes element

Maps

■ Implementation approaches

- **Two parallel arrays**
 - **Unsorted**
 - **Sorted**
- **Linked list**
- **Binary search tree**
- **Hash table**



■ Java Collections Framework

- **TreeMap** → uses red-black (balanced) tree
- **HashMap** → uses hash table

Maps

■ Complexity (average case)

	Put	Get	Remove
Unsorted array	$O(1)$	$O(n)$	$O(n)$
Sorted array	$O(n)$	$O(\log(n))$	$O(n)$
Linked list	$O(1)$	$O(n)$	$O(n)$
Binary search tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
Hash table	$O(1)$	$O(1)$	$O(1)$