Data Structures and Algorithms

XMUT-COMP 103 - 2026 T1

Traversing trees

Agatha Rachmat

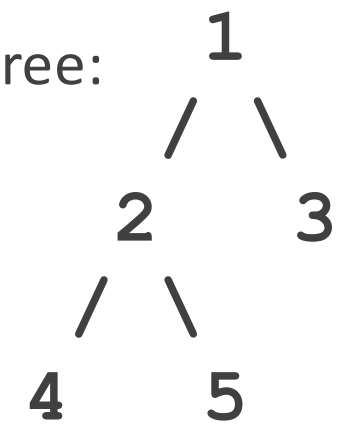
School of Engineering and Computer Science

Victoria University of Wellington

Review

1. Explain what *Traversing* is
2. What are the types of traversing?
3. Explain the differences between the types
4. What is Tree-structured data?

1. Why use a Queue in BFS?
2. Difference between BFS and DFS?
3. Where is BFS used?
4. Perform level-order traversal of this binary tree:



Traversing and returning

- Finding a single node or value to return:
  - In recursive traversal, must pass back the answer, all the way up the tree

There are 2 ways:

- Recursive
- Queue

Example 1: Traversing and returning

```
public class Person {
  String name;
  int yoB; // Year of Birth
  Person left;
  Person right;

  public Person(String name, int yoB) {
            this.name = name;
            this.yoB = yoB;
            this.left = null;
            this.right = null;
  }

  public Person getLeft() {
            return left;
  }

  public Person getRight() {
            return right;
  }

  public int getYoB(){
            return yoB;
  }
}
```

Traversing and returning: Recursive

```
Person findPerson(Person root, String targetName) {

        if (root == null)
                return null;

        if (root.name.equals(targetName)) // Check current node
                return root;

        // Search left subtree
        Person leftResult = findPerson(root.left, targetName);

        // If found on the left side
        if (leftResult != null)
        return leftResult;

        // Search right subtree
        return findPerson(root.right, targetName);
}
```

Traversing and returning: Queue

```
public static void main(String[] args) {

    Person root = new Person("John", 1980);

    root.left = new Person("Alice", 1995);
    root.right = new Person("Bob", 1992);

    root.left.left = new Person("Emma", 2000);
    root.left.right = new Person("David", 2003);

    Person result = findPerson(root, ”David");

    if (result != null)
        UI.println("Found: " + result.name + ", Year of Birth: " + result.yoB);
    else
        UI.println("Person not found");
}
```

Found: David, Year of Birth: 2003

Traversing and returning

The method uses **Depth-First Search (DFS)** recursion:

1.Check current node
2.Search left subtree
3.Search right subtree
4.Return the matching Person

Time Complexity

Case	Complexity
Best Case	O(1)
Worst Case	O(n)

Traversing and returning: Queue

```
import java.util.*;

Person findPerson(Person root, String targetName) {

   if (root == null)
      return null;

   Queue<Person> queue = new LinkedList<>(); // Queue for BFS traversal
   queue.offer(root);

   while (!queue.isEmpty()) {
      Person current = queue.poll();      // Remove front node

      if (current.name.equals(targetName)) // Check current person
         return current;

      if (current.left != null)                          // Add left child
         queue.offer(current.left);

      // Add right child
      if (current.right != null)
         queue.offer(current.right);
   }

   // Person not found
   return null;
}
```

Traversing and returning: Queue

```
public static void main(String[] args) {

    Person root = new Person("John", 1980);

    root.left = new Person("Alice", 1995);
    root.right = new Person("Bob", 1992);

    root.left.left = new Person("Emma", 2000);
    root.left.right = new Person("David", 2003);

    Person result = findPerson(root, "Emma");

    if (result != null)
        UI.println("Found: " + result.name + ", Year of Birth: " + result.yoB);
    else
        UI.println("Person not found");
}
```

```
Found: Emma, Year of Birth: 2000
```

Time Complexity: O(n)

Review

- Create a method to find a single node or value to return: