# ♦ Tree traversal algorithms in Java

Posted on October 18, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

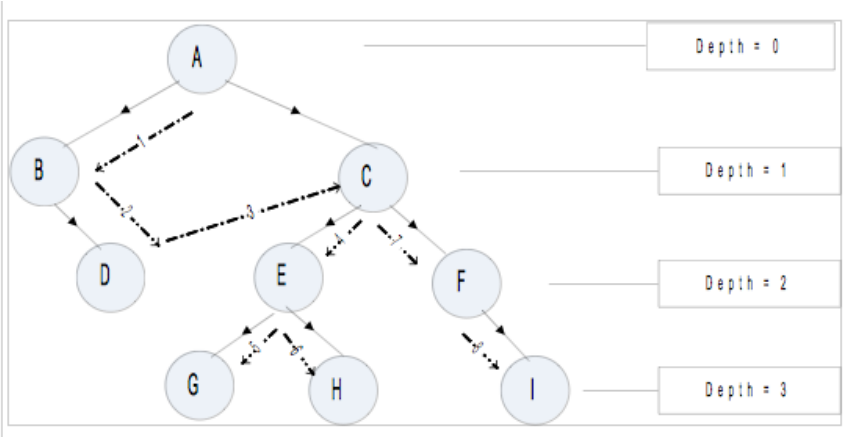
Q1. What are the different binary tree traversal mechanisms?

A1. **Traversing a tree** means visiting all the nodes of a tree in order. Many different binary tree algorithms involve traversals. For example, if you wish to count the number of employees in an organizational chart you must visit each node. If you wish to find the highest salary or add all the salaries of your employees by department, you must examine the value contained in each node. Report generation is an important task of Web based applications. A tree structure can be used to model the layout of nested reports and traverse the tree to produce nested results and offer aggregation functions.

There are two fundamentally different kinds of binary tree traversals – those that are **depth-first** and those that are **breadth-first**. There are three different types of depth-first traversals, which are **preorder**, **inorder**, and **postorder**.

There is only one kind of breadth-first traversal, which is the **level order traversal**. Trees can be traversed recursively or iteratively. Recursive traversal is the best known and most frequently used. Recursive algorithm uses method call stack in order to keep the state of the traversal for every level of a tree.

Q2. What is a preorder traversal, and can you give both recursive and iterative code examples?
A2. Preorder traversal gets its name from the fact that it visits the root first. In the case of a binary tree, the algorithm is → visit the root first, and then traverse the left subtree, and then traverse the right subtree.




**Prints**: ABDCEGHFI

In preorder traversal, each node is visited before any of its children. The code snippet for **recursion** is shown below.

```
1  //preorder traversal for a binary tree
2  public void traverse(Node node) {
3      //Exit condition for recursion
4      if (node == null) {
5            return;
6      }
7
8      System.out.println(node.getName( ));
9      traverse(node.getLeft( ));
10     traverse(node.getRight( ));
11
12 }
```

The above algorithm can be achieved **iteratively** without recursion as shown below by using a **Dequeue** (i.e. LIFO

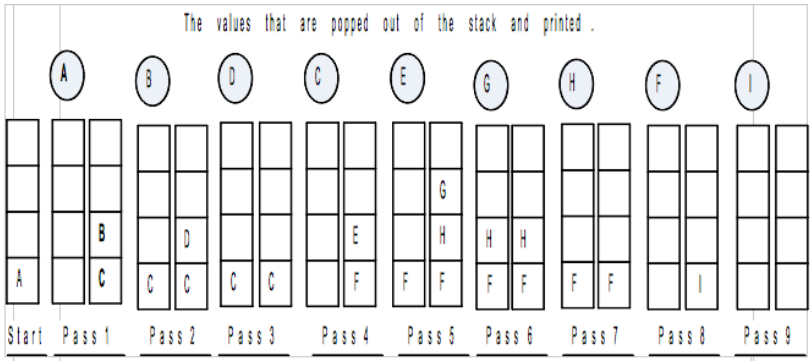


## 16 Technical Key Areas

using a double-ended queue).

```
1  public static void preorderIterative(Node node)
2          Deque<Node> s = new ArrayDeque<Node>(10)
3          s.push(node); // push the root node
4
5          while (!s.isEmpty( )) {
6              node = s.pop( );
7              System.out.print(node.getValue( ));
8
9              if (node.getRight( ) != null) {   //
10                 s.push(node.getRight( ));      /
11             }
12
13             if (node.getLeft( ) != null) {    //
14                 s.push(node.getLeft( ));       /
15             }
16         }
17 }
```

The diagram below shows how a double ended queue is used for traversal. Each pass within the while loop will have a value popped up (e.g. A), and its children pushed in (e.g. B and C).




preorder tree traversal

Q3. What is an **inorder traversal**, and can you give both recursive and iterative code examples?

A3. In a binary tree in-order traversal, left child tree is visited first, then visit the parent node, and then visit the right child tree.

## 80+ step by step Java Tutorials

open all | close all

- Setting up Tutorial (6)
- Tutorial - Diagnosis (2
- Akka Tutorial (9)
- Core Java Tutorials (2
- Hadoop & Spark Tuto
- JEE Tutorials (19)
- Scala Tutorials (1)
- Spring & HIbernate Tu
- Tools Tutorials (19)
- Other Tutorials (45)

## 100+ Java pre-interview coding tests

open all | close all

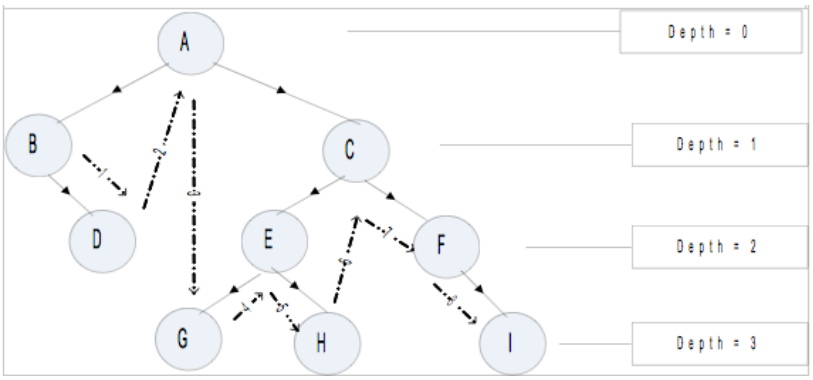
- Can you write code? (
- ♦ Complete the given
- Converting from A to I
- Designing your classe
- Java Data Structures
- Passing the unit tests
- What is wrong with th
- Writing Code Home A
- Written Test Core Jav
- Written Test JEE (1)

## How good are your .....?

open all | close all

- Career Making Know-

Job Hunting & Resum




inorder tree traversal

**Prints**: BDAGEHCFI

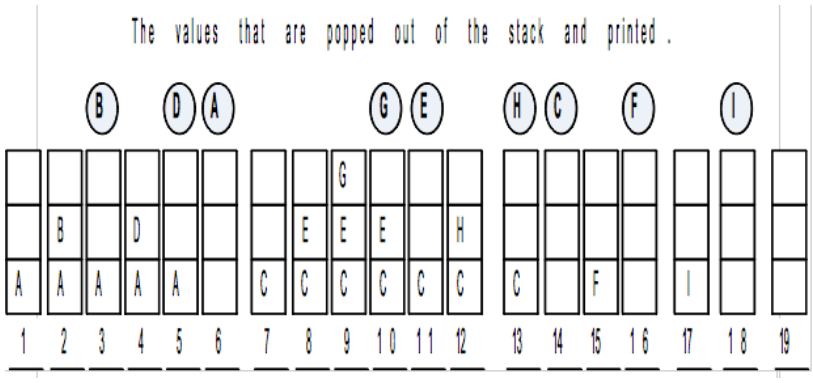
The **recursive** approach is shown below.

```
1  //inorder traversal for a binary tree
2  public void traverse(Node node) {
3      //Exit condition for recursion
4      if (node == null) {
5         return;
6      }
7
8      traverse(node.getLeft( ));
9      System.out.println(node.getName( ));
10     traverse(node.getRight( ));
11 }
```

The **iterative** approach is shown below using a LIFO approach:

```
1  public static void inorderIterative(Node node) {
2          Deque<Node> s = new ArrayDeque<Node>(10)
3          while (!s.isEmpty() || null != node) {
4              if (null != node) {
5                  s.push(node);
6                  node = node.left;
7              } else {
8                  node = s.pop();
9                  System.out.print(node.getValue()
10                 node = node.right;
11             }
12         }
13 }
```

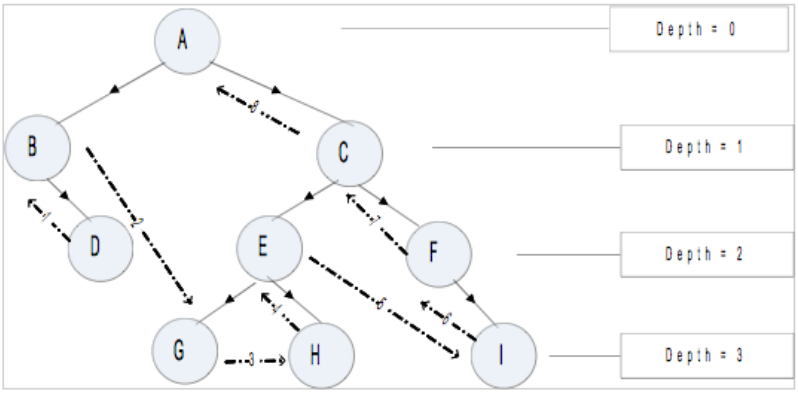
The diagram below shows how a double ended queue is used for traversal. All depends on how the elements are popped in and popped out.




inorder iterative traversal

Q4. What is an **postorder traversal**, and can you give both recursive and iterative code examples?

A4. In a binary tree postorder traversal, the left and right subtrees are visited before the parent node.

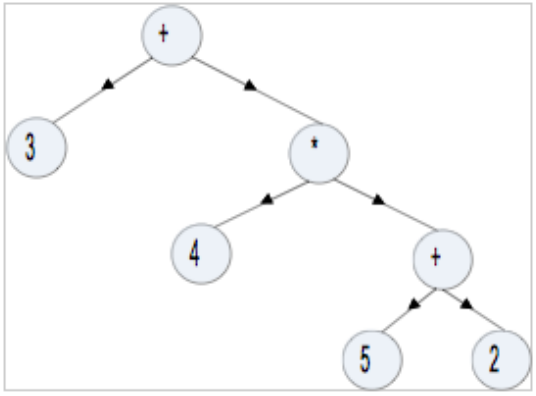



postorder traversal

**Prints:** DBGHEIFCA

```
1  //postorder traversal for a binary tree
2  public void traverse(Node node) {
3      //Exit condition for recursion
4      if (node == null) {
5          return;
6      }
7
8      traverse(node.getLeft( ));
9      traverse(node.getRight( ));
10     System.out.println(node.getName( ));
11 }
```

Q5. What is an **expression tree**?
A5. An expression tree is a binary tree that represents a mathematical expression. It is shown below to demonstrate tree traversals discussed above.




Mathematical expression tree

**Preorder traversal**: + 3 * 4 + 52
**Inorder traversal**: 3 + 4 * 5 + 2
**Postorder traversal**: 3452+*+

Q6. Can you give some real life examples of these tree traversals?
A6. **Preorder traversal** can be used to create or clone an existing tree. The parent needs to exist before the children can be added. Evaluation of expressions in prefix notation and processing of the abstract syntax trees by compilers (e.g. JavaCC, ANTLR, etc) are based on preorder traversal. The DefaultMutableTreeNode is a general purpose tree data structure in the package javax.swing.tree. The getNextNode( ) method in this class returns the node that follows a preorder traversal. A JavaScript method like getElementsByTagName(…) could use a preorder traversal to return the elements from a HTML DOM (Document Object Model) tree. It can also be used in your web pages for the site map, menus, and the bread crumb navigation due to its simplicity of retrieving parents and children.

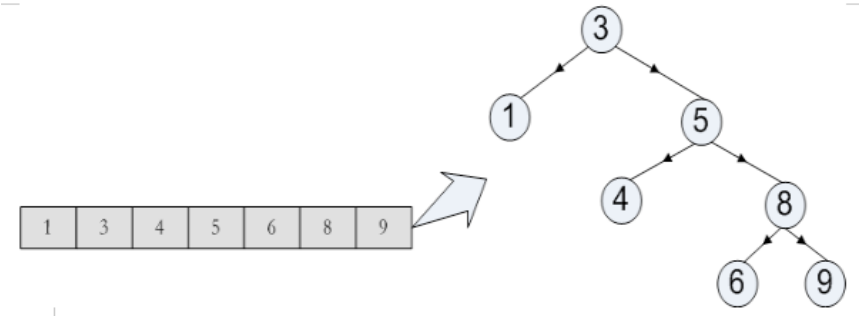
**Postorder traversal** is used for evaluating of expressions in post-fix, and by the Reverse Polish Notation (RPN), which is

used by the compilers. For example, a machine language will require a notation like 2 3 +. A postorder is required for destroying a tree. All the children needs to be destroyed before the parent can be destroyed.

```
1 public void destroy(Node node) {
2       if (node == null) {
3       return;                          // exit cond
4       }
5       destroy(node.left)
6       destroy(node.right)
7
8      freeCurrentNode(node)
9 }
```

Any situations where sub tasks need to be performed before the parent tasks. For example, nested pop up menus where child menus need to be closed before the parent menus can be closed. In a build process or a Gantt chart, sub tasks or processes need to be completed before the parent tasks or processes can be completed.

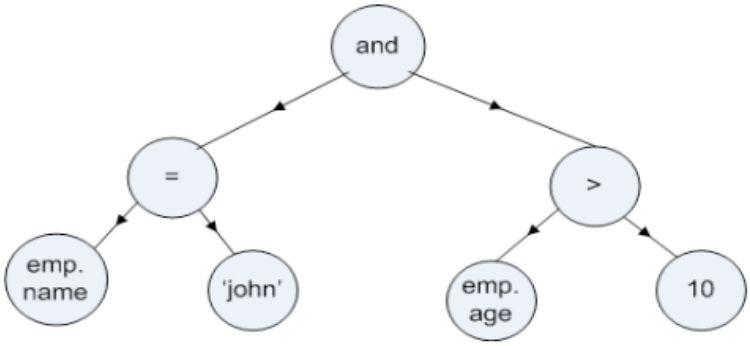
**Inorder** traversal can be used to search for an item in a balanced binary tree using the binary search algorithm. You can also print all of their data in alphanumeric order.




The WHERE clause SQL expressions can be parsed using inorder tree traversal. For example the SQL expression

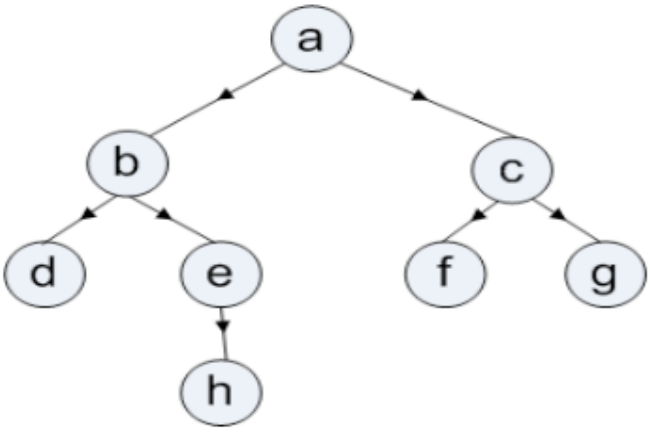
```
1 SELECT * from employees WHERE  emp.name = 'john'
```

can be parsed for WHERE clause using an in-order traversal as shown below:




Q7. What are the different searching methodologies for trees that may not have any particular ordering? Which approach will you use if you are likely to find the element near the top of the tree? Which approach will you be using if you are likely to find the element near the bottom of the tree?

A7. There are 2 types of searches known as **BFS** and **DFS**.




**Breadth First Search (BFS) aka level-order traversal.**

The BFS searches through a tree from the root and moves from nodes left to right at each level (i.e depth 0, depth 1, etc) until either the search element has been found or all the nodes have been looked at. As per the diagram, it will be searched in the order of a, b, c, d, e, f, g, and h. The preorder traversal uses a stack, and the BFS uses a queue. The queue (FIFO) and stack (LIFO) are pretty much opposite. The LinkedList class can be used as a queue for storing children on the same level. The code snippet below demonstrates the **BFS for a binary tree**.

```
1  public boolean bfsSearch(Node root, String searc
2      if (root == null) {
3          return false;
4      }
5
6      Queue<Node> q = new LinkedList<Node>( );
7      q.add(root);
8
9      Node tmp;
10
11     while (q.size( ) > 0) {
12         tmp = q.remove( );
13
14         if (tmp.getName( ).equals(search)) {
15             return true;
16         }
17
18         if (tmp.getLeft( ) != null) {
19             q.add(tmp.getLeft( ));
20         }
21
22          if (tmp.getRight( ) != null) {
23             q.add(tmp.getRight( ));
24         }
25      }
26
27      return false;
28 }
```

**Depth First Searching (DFS)**

A DFS searches through a tree starting at the root, and goes straight down a single branch until a leaf is reached. Then, it repeats the same process with its nearest ancestor that has not been searched through yet. As per the diagram, it will be searched in the order of a, b, d, h, e, c, f, and g.

The code snippet below demonstrates the DFS for a binary tree.

```
1  public boolean dfsSearch(Node node, String searc
2      //exit condition for recursion
3      if (node == null) {
4          return false;
5      }
6
7      //exit condition for recursion
8      if (node.getName( ).equals(search)) {
9          return true;
10     }
11
12     //recursive calls
13     return dfsSearch(node.getLeft( ), search) ||
14 }
```

# Popular Posts

♦ 11 Spring boot interview questions & answers

**823 views**

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

**765 views**

18 Java scenarios based interview Questions and Answers

**399 views**

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

**388 views**

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

**295 views**

♦ 7 Java debugging interview questions & answers

**293 views**

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

**285 views**

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

**279 views**

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

**239 views**

001B: ♦ Java architecture & design concepts interview questions & answers

**201 views**

Bio | Latest Posts

## Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in

2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

**About** Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Searching algorithms in Java

♥ 1. Setting up Java, Maven, and eclipse ›

**Posted in** Algorithms, member-paid, Tree structure

# Leave a Reply

Logged in as geethika. Log out?

**Comment**

Post Comment

# Empowers you to open more doors, and fast-track

### Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

### Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

# Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

# © Disclaimer