N- Ary Tree Implementation

FNU Anirudh (Author)
Indiana University
Bloomington
MS in Data Science
aanirudh@iu.edu

Venkata Prudhvi Raj Indana (Author)
Indiana University
Bloomington
MS in Data Science
vindana@iu.edu

Abstract

The goal of this project is to implement N- ary tree in JAVA. N- ary trees are trees in which each node may have up to n children. N-array tree is also called as K-ary tree. A binary tree is special case of K-ary tree with k=2. In this project we plan to implement.

The primary focus of this paper is to implement N-ary tree data structure and then perform N-array tree traversal and then to find the total number of nodes in the tree.

1. Introduction

A data structure is a particular way of organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services. Usually, efficient data structures are key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Data structures can be used to organize the storage and retrieval of information stored in both main memory and in secondary memory. There are numerous types of data structures built upon similar data types: 1)Array, 2)Associative array, 3) Record, 4) Union, 5) Union, 6) Tagged Union, 7) Set, 8) Graph and Tree, 9) Class.

We will be limiting our experiment to trees as far as scope of this paper is concerned, more specifically n-ary trees.

A tree is a (possibly non-linear) data structure made up of nodes or vertices and edges without having any cycle. The tree with no nodes is called the null or empty tree. A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a hierarchy.

1.1 Terminologies used in Trees

Root	The top node in a tree
Child	A node directly connected to another node when moving away from the Root.
Parent	The converse notion of a child.
Siblings	Nodes with the same parent.
Descendant	A node reachable by repeated proceeding from parent to child.
Ancestor	A node reachable by repeated proceeding from child to parent.
Leaf	A node with no children

Internal node	A node with at least one child
External node	A node with no children
Degree	Number of sub trees of a node
Edge	Connection between one node to another
Path	A sequence of nodes and edges connecting a node with a descendant.
Level	The level of a node is defined by 1 + (the number of connections between the node and the root).
Height of node	The height of a node is the number of edges on the longest path between that node and a leaf.
Height of tree	The height of a tree is the height of its root node
Depth	The depth of a node is the number of edges from the node to the tree's root node.
Forest	A forest is a set of n ≥ 0 disjoint trees

N-ary tree: - is a rooted tree in which each node has no more than k children. It is also sometimes known as a k-way tree, an N-ary tree, or an M-ary tree. A binary tree is the special case where k=2.

1.2 Types of N-ary trees

- A **full n-ary tree** is a n-ary tree where within each level every node has either *0* or *k* children.
- A **perfect n-ary tree** is a full n-ary tree in which all leaf nodes are at the same depth.
- A **complete n-ary tree** is a n-ary tree which is maximally space efficient. It must be completely filled on every level except for the last level. However, if the last level is not complete, then all nodes of the tree must be "as far left as possible".

1.3 Properties of N-ary tree

- For a k-ary tree with height *h*, the upper bound for the maximum number of leaves is k^h.
- The height *h* of a **k-ary tree** does not include the root node, with a tree containing only a root node having a height of 0.
- The total number of nodes in a perfect k-ary tree is , while the height h is

$$\frac{k^{h+1}-1}{k-1}$$

Note: A tree containing only one node is taken to be of height 0 for this formula to be applicable.

1.4 Applications of N-ary Tree

- We can use N-ary trees in applications like file directories.
- We can use N-ary trees in dictionaries.
- We can use N-ary trees in family hierarchy.

- We can also use it organization hierarchy.

2. Operations and Implementation

N-ary tree data structure:-

A node generally has n-1 keys and n children. Each node has alternating sub-tree pointers and keys:
sub-tree | key | sub-tree | key | ... | key | sub_tree

i. All keys in a sub-tree to the left of a key are smaller than it.
ii. All keys in the node between two keys are between those two keys.
iii. All keys in a sub-tree to the right of a key are greater than it.
iv. This is the "standard" recursive part of the definition.

2.1 Pseudo code:-

N-ary tree consists of a root
containing j (1<=j<m) keys, kj, and
a set of sub-trees, Ti, (i = 0..j), such that

i. if k is a key in T0, then k <= k1
ii. if k is a key in Ti (0<i<j),
then ki <= k <= ki+1
iii. if k is a key in Tj, then k > kj and
iv. all Ti are nonempty m-way search trees or all Ti are empty

2.2 Tree traversal

In computer science, tree traversal (also known as tree search) is a form of graph traversal and refers to the process of visiting (checking and/or updating) each node in a tree data structure, exactly once. Such traversals are classified by the order in which the nodes are visited. The following algorithms are described for an N-ary tree. Tree structures can be traversed in many different ways. Starting at the root of a binary tree, there are three main steps that can be performed and the order in which they are performed defines the traversal type. These steps (in no particular order) are: performing an action on the current node (referred to as "visiting" the node), traversing to the left child, and traversing to the right child.

N-ary trees can be traversed in pre-order or post-order. These searches are referred to as depth-first search (DFS), as the search tree is deepened as much as possible on each child before going to the next sibling. For a n-ary tree, they are defined as display operations recursively at each node, starting with the root, whose algorithm is as follows:

The general recursive pattern for traversing is explained in detail below.

a) **Pre-order**

Take an empty list of type Node. Add the root to the empty list and then traverse all its Childs recursively adding them to the list in the order they are processed and returning the list. Once we are done with the recursion we will have a list having elements in preorder

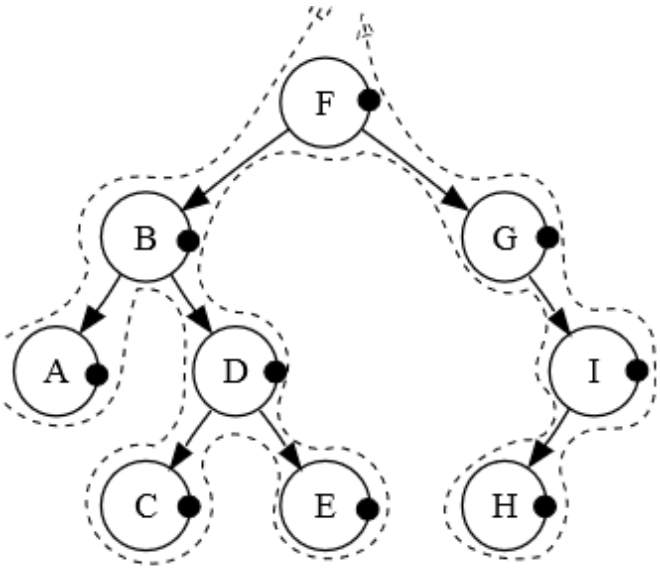

Image showing Post-order traversal in a binary tree: A, C, E, D, B, H, I, G, F

Pseudo Code

1. Preorder_traversal(struct Node *root)
// base condition for recursion
// if tree/sub-tree is empty, return and exit
2. if(root == NULL) return
3 Print data present in root
4. Preorder_traversal(root->left) // Visit left subtree
5. Preorder_traversal(root->right) // Visit right subtree

a) Post-order

Take an empty list of type Node. Traverse all its Childs recursively and once we are at the end of the recursion adding them to the list in the order they are being returned and returning the list. Once we are done with the recursion we will have a list having elements in post-order.

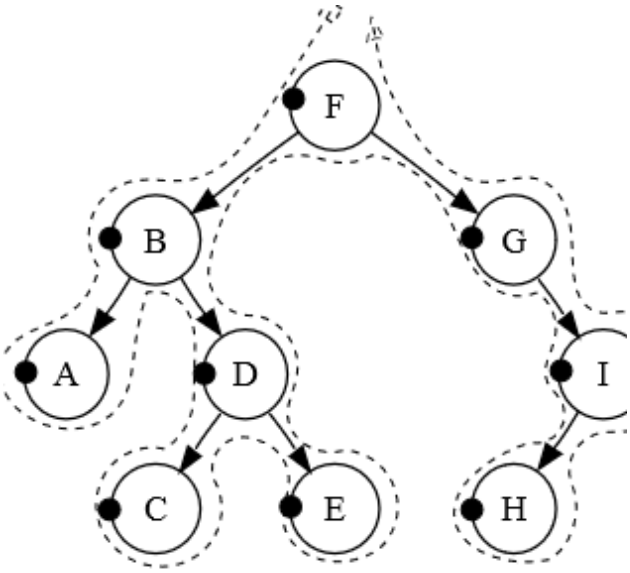

Image showing Pre-order traversal in a binary tree: F, B, A, D, C, E, G, I, H.

Pseudo Code:-

1. Postorder_traversal(Node *root)
2. if(root == NULL) return;
3. Postorder_traversal(root->left) // Visit left subtree
4. Postorder_traversal(root->right) // Visit right subtree
5. Print data present in root

Search in an N-ary tree:-

Use one the above traversal methods and access data present in all the nodes. Compare data present in children with value given, In case we find a match return the node, else return Null.

Number of nodes in an N-ary tree:-

Initialize a counter to 0, choose one the above traversal method and access Childs of each node by recursion by taking a counter and incrementing it for each child accessed and returning it back. At the end of the recursion we should have the count of nodes in a N-ary tree.

Pseudo Code

1. getMaxNodes()
2. return getNOD(root) + 1;
3. getNOD(Node<T> node)
4. int n = node.getChildren().size();
5. for each child of the parent node
6. n += getNOD(child);
7. return n;

Conclusion

N-ary tree have definitely proven to be one of the useful data structures in computer science due to their numerous applications. They can be used for directories, family hierarchy, and organization hierarchy and in other areas.

We implemented basic version of one of the variants of higher order trees and performed operations like tree- traversal, finding maximum number of nodes and searching in a tree.

This paper covers implementation of n-ary tree and basic operations like searching, finding maximum number of nodes and tree- traversal. When compared with binary tree , n-ary tree

are not good choice. Binary trees are less complicated and easy to implement. N-ary trees have advantage of storing multiple childs. A B-tree which variant of n-ary tree has the advantage that is all the leaves are linked together sequentially, the entire tree may be scanned without visiting the higher nodes at all. A variation of the B-tree, known as a B+-tree considers all the keys in nodes except the leaves as dummies. For this paper we limit our scope to n-ary trees.

References

1. http://www2.cs.uregina.ca/~beattieb/CS340/notes/09%20-%20Non-Binary%20Trees.pdf
2. Automatic Construction of N-ary Tree Based Taxonomies by Kunal Punera, Suju Rajan, Joydeep Ghosh
3. Lexicographic Listing and Ranking of r-ary Trees by M.C. ER
4. https://en.wikipedia.org/wiki/K-ary_tree
5. https://www.cs.cmu.edu/~pattis/15-1XX/15-200/lectures/specialtrees/
6. http://www.geeksforgeeks.org/serialize-deserialize-n-ary-tree/
7. N-ARY TREES CLASSIFIER by Duarte Duque, Henrique Santos, Paulo Cortez
8. https://en.wikipedia.org/wiki/Tree_traversal
9. https://www.cs.auckland.ac.nz/~jmor159/PLDS210/n_ary_trees.html
10. https://www.hackerrank.com/domains/data-structures/trees