Laboratory 6: Binary Search Tree algorithms

CSC205A Data structures and Algorithms Laboratory B. Tech. 2015

Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: vaishali.cs.et@msruas.ac.in
Tel: +91-80-4906-5555 (2212) WWW: www.msruas.ac.in

Binary Search Tree algorithms

Introduction and Purpose of Experiment

- Binary tree algorithms such as Breadth First Search (BFS), Depth First Search (DFS) are useful tree traversal techniques used in many applications.
- This experiment introduces the binary search algorithms and its applications.

Aim:

- To design and implement the algorithms for Depth First Search Traversal (Postorder) and Breadth First Search Traversal (Level Order) using both recursive and non-recursive way to traverse a given BST.

Binary Search Tree algorithms

Objectives:

At the end of this lab, the student will be able to

- Design and Implement BFS tree algorithm
- Apply BFS for traversal and search
- Design and Implement DFS(Postorder) tree algorithm
- Apply DFS for traversal and search
- Compare the efficiency of contemporary algorithms in both DFS and BFS

Binary Search Tree algorithms

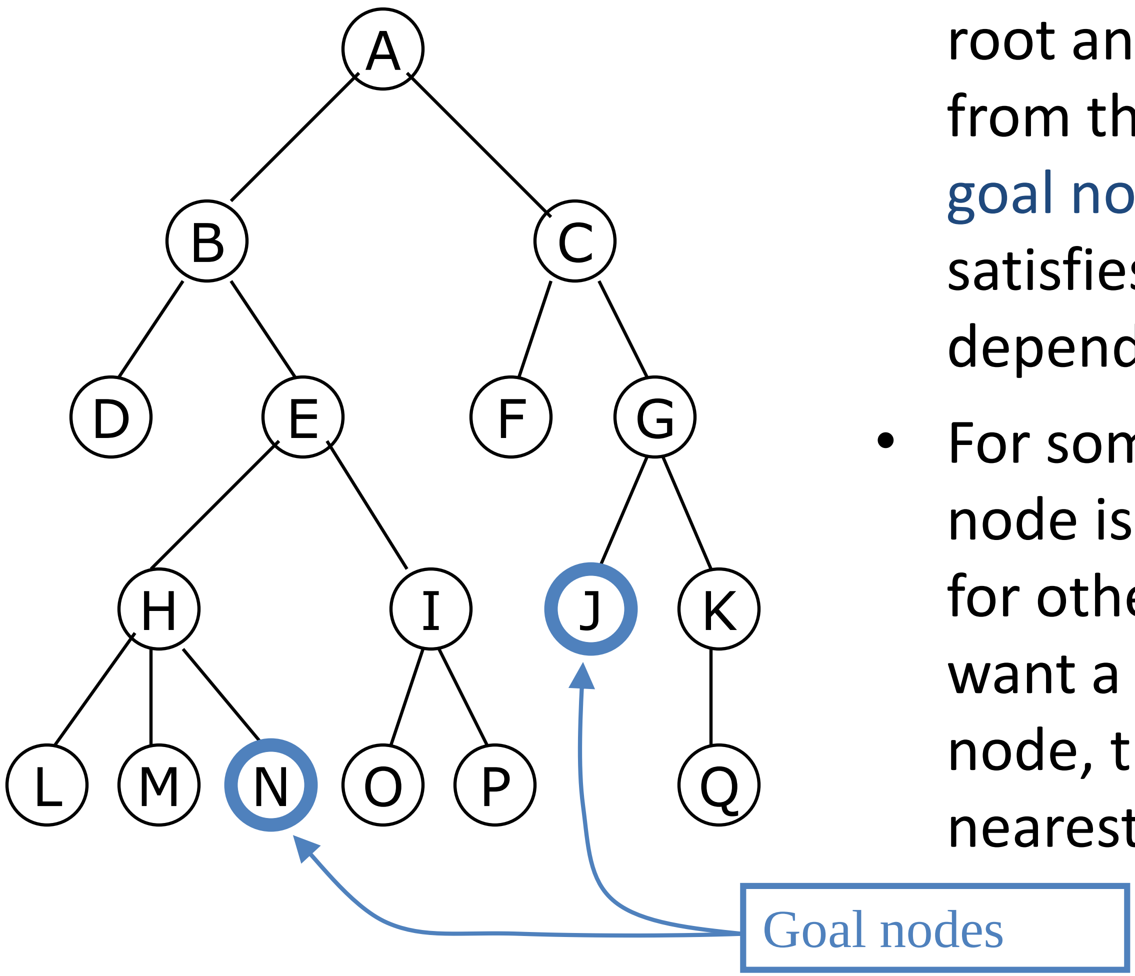
Theory:

- Tree traversal refers to the process of visiting (processing, printing, updating) each node exactly once.
- In linear data structures, all the elements are arranged in a sequence hence there is only one possible traversal. Trees are non linear data structures, the elements are in different levels, exhibiting the hierarchy among them. Hence a tree can be traversed in many ways.
- BST can be traversed in many ways such as Depth First Search Traversal which includes preorder, inorder, postorder and Breadth First Search Traversal
- BST traversals are significant due its applications

Tree Traversal Algorithms

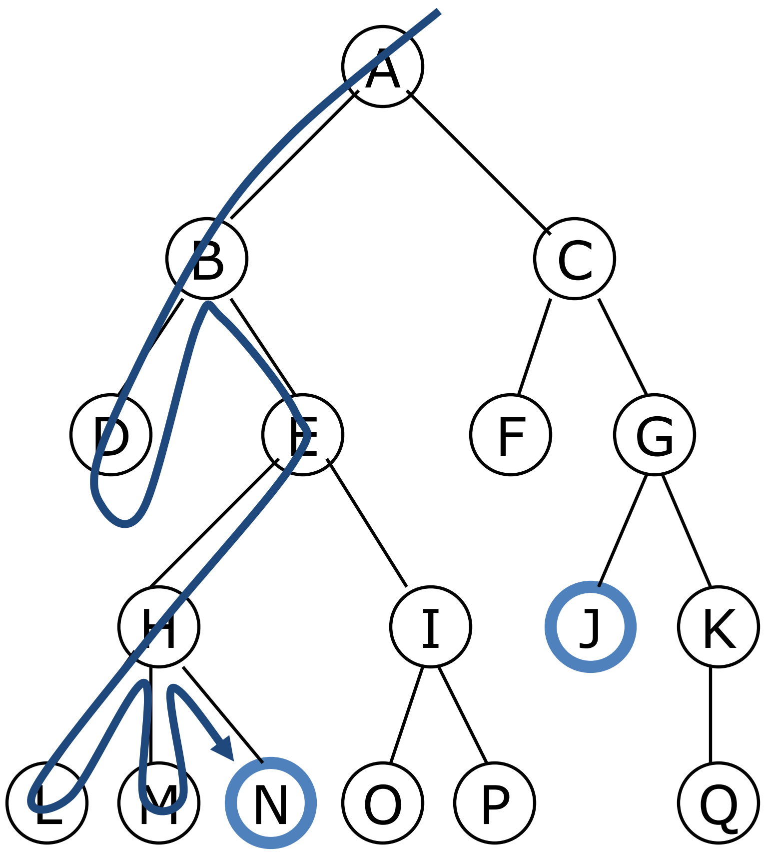
- Traversing a tree means to visit each of its nodes exactly one in particular order
  - Many traversal algorithms are known
  - Depth-First Search (DFS)
    - Visit node's successors first
    - Usually implemented by recursion
  - Breadth-First Search (BFS)
    - Nearest nodes visited first
    - Implemented by a queue

Tree searches



- A tree search starts at the root and explores nodes from there, looking for a goal node (a node that satisfies certain conditions, depending on the problem)
- For some problems, any goal node is acceptable (N or J); for other problems, you want a minimum-depth goal node, that is, a goal node nearest the root (only J)

Depth-first searching



- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path
- For example, after searching A, then B, then D, the search backtracks and tries another path from B
- Node are explored in the order A B D E H L M N I O P C F G J K Q
- N will be found before J

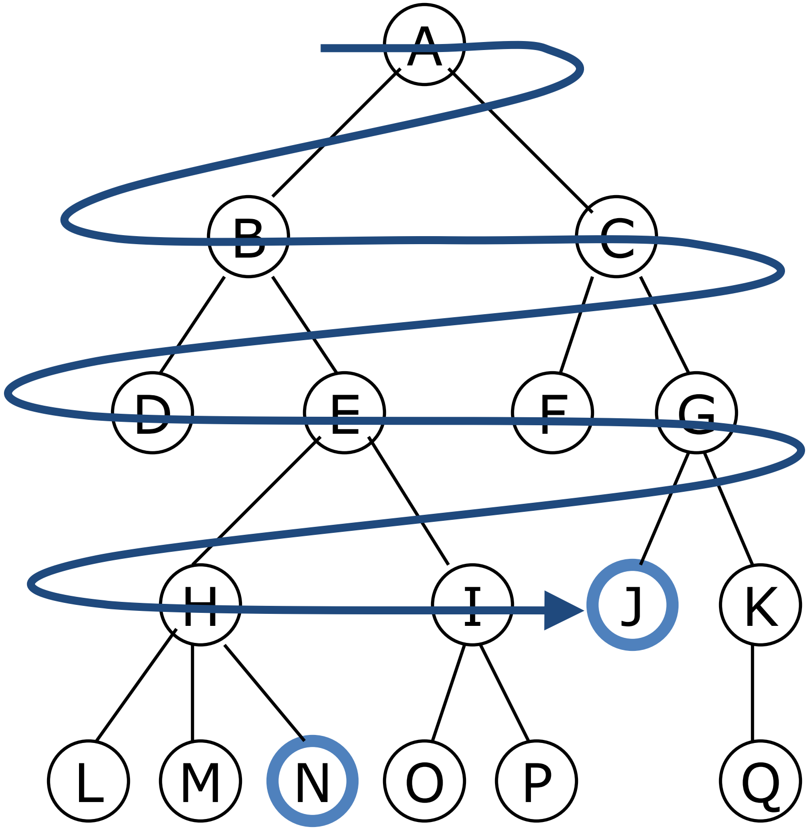
How to do depth-first searching

- ```
  Put the root node on a stack;
  while (stack is not empty) {
      remove a node from the stack;
      if (node is a goal node) return success;
      put all children of node onto the stack;
  }
  return failure;
  ```
- At each step, the stack contains some nodes from each of a number of levels
  - The size of stack that is required depends on the branching factor b
  - While searching level n, the stack contains approximately b*n nodes
- When this method succeeds, it doesn't give the path
  ```

Recursive depth-first search

- search(node):
    if node is a goal, $\left\{\text{print node and } \quad \text{return success;}\right\}$
    for each child c of node {
        if search(c) is successful, $\left\{\text{print c and } \quad \text{return success;}\right\}$
    }
    return failure;
- The (implicit) stack contains only the nodes on a path from the root to a goal
  - The stack only needs to be large enough to hold the deepest search path
  - When a solution is found, the path is on the (implicit) stack, and can be extracted as the recursion "unwinds"

Breadth-first searching



- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Node are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

How to do breadth-first searching

- ```
  Put the root node on a queue;
  while (queue is not empty) {
      remove a node from the queue;
      if (node is a goal node) return success;
      put all children of node onto the queue;
  }
  return failure;
  ```
- Just before starting to explore level n, the queue holds *all* the nodes at level n-1
- In a typical tree, the number of nodes at each level increases *exponentially* with the depth
- Memory requirements may be infeasible
- When this method succeeds, it doesn't give the path
- There is *no* "recursive" breadth-first search equivalent to recursive depth-first search
  ```

Comparison of algorithms

- Depth-first searching:
  - Put the root node on a stack;
    ```
    while (stack is not empty) {
        remove a node from the stack;
        if (node is a goal node) return success;
        put all children of node onto the stack;
    }
    return failure;
    ```
- Breadth-first searching:
  - Put the root node on a queue;
    ```
    while (queue is not empty) {
        remove a node from the queue;
        if (node is a goal node) return success;
        put all children of node onto the queue;
    }
    return failure;
    ```

Binary Search Tree algorithms

Experimental Procedure:

- Analyse the problem statement
- Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
- Implement the algorithm in C language
- Compile the C program
- Design test cases and test the implemented program
- Document the Results
- Analyse and discuss the outcomes of your experiment

Binary Search Tree algorithms

Exercises

Design, develop algorithms and write C program to traverse the given BST using DFS(Post order) and BFS(Level Order) using recursion and non-recursive way(Queues).

- Design the test cases to test the implemented C program and verify against expected values.
- Analyse the efficiency of both the algorithms.
- Describe your learning along with the limitations of both, if any. Suggest how these can be overcome.

Binary Search Tree algorithms

Approach:

1. Define the structure of the BST in the header file.
2. Define the required functions and its signature in the header file
   - Creating a node
   - Inserting a node in to BST and so on
   - Postorder traversal
   - Separate functions for Level Order traversal using recursion and non-recursive way(using Queues)
3. Implement the function declared in the header file in a separate C-Source File.
4. In the main function, write the appropriate logic for reading input from user and calling the required functions and printing the results.

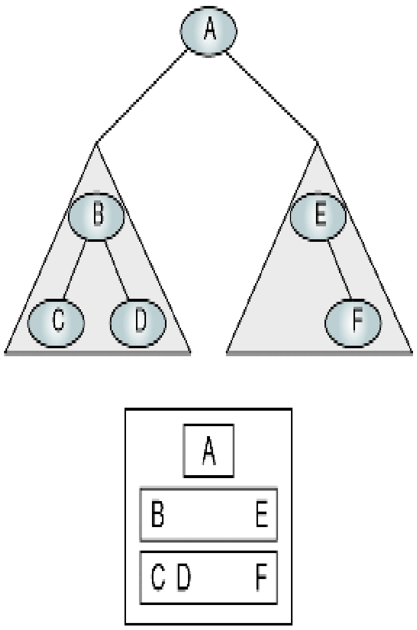
Binary Search Tree algorithms

Algorithm for Post Order raversal

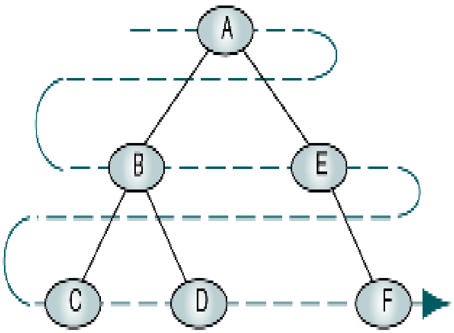
```
Algorithm postOrder (root)
Traverse a binary tree in left-right-node sequence.
   Pre   root is the entry node of a tree or subtree
   Post  each node has been processed in order
1 if (root is not null)
   1   postOrder (left subtree)
   2   postOrder (right subtree)
   3   process (root)
2 end if
end postOrder
```

Binary Search Tree algorithms

Level Order (Breadth First) Traversal:



(a) Processing order



(b) "Walking" order

```
/*Function to print level order traversal of tree*/
printLevelorder(tree)
for d = 1 to height(tree)
   printGivenLevel(tree, d);

/*Function to print all nodes at a given level*/
printGivenLevel(tree, level)
if tree is NULL then return;
if level is 1, then
    print(tree->data);
else if level greater than 1, then
    printGivenLevel(tree->left, level-1);
    printGivenLevel(tree->right, level-1);
```

Level Order : A, B, E, C, D, F

Results and Presentations

- Calculations/Computations/Algorithms

  The calculations/computations/algorithms involved in each program has to be presented

- Presentation of Results

  The results for all the valid and invalid cases have to be presented

- Analysis and Discussions

  how the data is manipulated or transformed, what are the key operations involved. Errors encounters and how they are resolved.

- Conclusions

  Summary

Comments

- Limitations of Experiments

  Outline the loopholes in the program, data structures or solution approach.

- Limitations of Results

  Present the test cases; justify if the program is tested correctly considering all the outcomes. Mention what is not tested, if any.

- Learning happened

  What is the overall learning happened

- Conclusions

  Summary

References

- Gilberg, R. F., and Forouzan, B. A. (2007): A Pseudocode Approach With C, 2nd edn. Cengage Learning
- The algorithm for recursive level order traversal is taken from:

  http://www.geeksforgeeks.org/level-order-tree-traversal/