CS32 Discussion
Week 8

Muhao Chen
muhaochen@ucla.edu
http://yellowstone.cs.ucla.edu/~muhao/
Outline

• Binary Trees
• Binary Search Trees
Tree: Definitions

- **root**
- **node** \(\rightarrow\) **link (edge)**
- **parent**
- **children**
- **siblings**
- **leaves**
Tree: Definitions

- **root**
- **node** → **link (edge)**
- **parent**
- **children**
- **siblings**
- **leaves**
- **subtree**
- **height**

No loop!
Bound on # of edges

How many edges should there be in a tree of n nodes?

n - 1
Binary Trees

No node has more than 2 children (left child + right child).
Binary Trees

How many nodes can a binary tree of height $h$ have?
(one with max. # of nodes == full binary tree)

$2^0 + 2^1 + \ldots + 2^{h-1} = 2^h - 1$
Tree is a data structure!

• For every data structure we need to know:
  – how to **insert** a node,
  – how to **remove** a node,
  – **search** for a node

• and (for tree only)
  – how to traverse the tree

```c
struct Node {
    ItemType val;
    Node* left;
    Node* right;
};
```
Three Methods of Traversal

void preorder(const Node *node) {
    if (node == NULL) return;
    cout << node->val << " ";
    preorder(node->left);
    preorder(node->right);
}

void inorder(const Node *node) {
    if (node == NULL) return;
    inorder(node->left);
    cout << node->val << " ";
    inorder(node->right);
}

void postorder(const Node *node) {
    if (node == NULL) return;
    postorder(node->left);
    postorder(node->right);
    cout << node->val << " ";
}
Level Order Traversal (Non-recursive)

```cpp
void level_order(const Node *root) {
    queue<Node*> q;
    q.push(root);
    while(!q.empty()) {
        Node *cur = q.first();
        q.pop();
        cout << cur->val;
        if (cur->left) q.push(left);
        if (cur->right) q.push(right);
    }
}
```
Traversal Orders

Pre-order DFS

In-order DFS

Post-order DFS

Level-order BFS
Problems about traversing binary trees that usually appear in your test

• Traverse and record some information.
  • What’s the height of a tree?
  • Is a binary tree balanced?

• Traverse and modify the tree.
  • Invert a binary tree.
int treeHeight(const Node *node)
{
    if (node == NULL)
        return 0;

    int leftHeight = treeHeight(node->left);
    int rightHeight = treeHeight(node->right);

    if (leftHeight > rightHeight)
        return leftHeight + 1;
    else
        return rightHeight + 1;
}
Invert a Binary Tree

Story of Max Howell, who invented Homebrew (a package management software widely used in Google Inc. and Apple Inc.), and failed the interview by Google because he couldn’t revert a binary tree.

Max Howell
@mxcl

Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.
Invert a Binary Tree

TreeNode* invertTree(TreeNode* root) {
    if (root) {
        std::swap(root->left, root->right);
        invertTree(root->left);
        invertTree(root->right);
    }
    return root;
}
Invert a Binary Tree

TreeNode* invertTree(TreeNode* root) {
  if (!root) return root;
  std::queue<TreeNode*> q;
  stk.push(root);

  while (!stk.empty()) {
    TreeNode* p = q.top();
    q.pop();
    std::swap(p->left, p->right);
    if (p->left) q.push(p->left);
    if (p->right) q.push(p->right);
  }
  return root;
}
Binary Search Tree
Binary Search Tree

- At all nodes:
  - All nodes in the left subtree have smaller values than the current node’s value
  - All nodes in the right subtree have larger values than the current node’s value
- Which traversal method should you use to:
  - print values in the increasing order?
  - print values in the decreasing order?
void insert(Node* &node, ItemType newVal)
{
    if (node == NULL)
    {
        node = new Node;
        node->val = newVal;
        node->left = node->right = NULL;
    }

    if (node->val > newVal)
        insert(node->left, newVal);
    else
        insert(node->right, newVal);
}
Insert

- **Average** time complexity?
  - as many steps as the height of the tree
  - full tree: \( n = 2^{h+1} - 1 \approx 2^{h+1} \) nodes
  - \( h \approx \log_2 n - 1 \)

- Roughly, it takes \( O(\log N) \).
Search

Node* search(const Node *node, ItemType value) {
    if (node == NULL)
        return NULL;

    if (node->val == value)
        return node;
    else if (node->val > value)
        return search(node->left, value);
    else
        return search(node->right, value);
}
Removal

- A little tricky!
- General strategy:
  - Find a replacement.
  - Delete the node.
  - Replace.
- Case-by-case analysis
  - Case 1: the node is a leaf (easy)
  - Case 2: the node has one child
  - Case 3: the node has two children
Case 3

Use in-order traversal to identify these nodes
findMax

ItemType findMax(const Node *node)
{
}

FindMax

ItemType findMax(Node *node) {
    if (node -> right == NULL) return node -> val;
    return findMax(node -> right);
}
FindMin

ItemType findMin(Node *node) {
    if (node -> left == NULL) return node->val;
    return findMax(node -> left);
}

bool valid(const Node *node)
{

  // At all nodes:
  // - All nodes in the left subtree have smaller values than the current node’s value
  // - All nodes in the right subtree have larger values than the current node’s value

}
bool valid(const Node *node) {
    if (node == NULL)
        return true;

    if (node->left != NULL && findMax(node->left) > node->val)
        return false;

    if (node->right != NULL && findMin(node->right) < node->val)
        return false;

    return valid(node->left) && valid(node->right);
}
Other Representative Trees

- B+-Tree (CS143)
- R-Tree (Spatial Index Tree)
- Quad-tree
Bugs in your software are actually special features :)

```php
if ($thirsty==TRUE)
{
}
else
{
}
```