GeekBand 极客班

互联网人才 + 油站!

GeekBand 极客班

www.geekband.com

4. Binary Tree

GeekBand 极客班

大纲

1. 树的概念
2. 树的深度遍历
3. 分治算法
4. 宽度优先
5. 二叉搜索树
6. **Trie Tree**

GeekBand 极客班

树的概念

树是一种能够分层储存数据的重要数据结构，树中的每个元素被称为树的节点，每个节点有若干个指针指向子节点。从节点的角度来看，树是由唯一的起始节点引出的节点集合。这个起始结点称为根(root)。树中节点的子树数目称为节点的度(degree)。

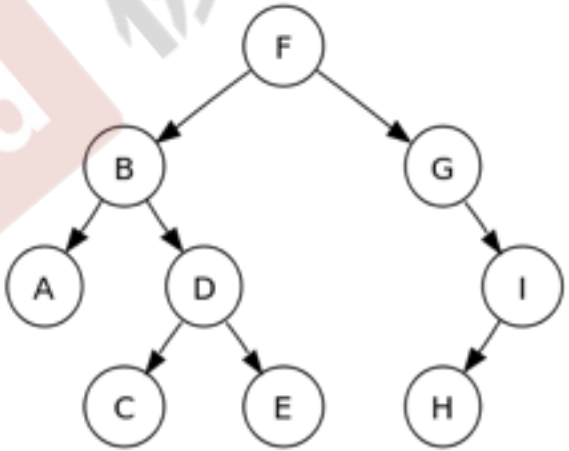
在面试中，关于树的面试问题非常常见，尤其是关于二叉树，二叉搜索树的问题。

GeekBand 极客班

二叉树

二叉树，是指对于树中的每个节点而言，至多有左右两个子节点，即任意节点的度小于等于2

二叉树的第i层至多有 2^{i-1}
深度为k的二叉树至多有 2^k-1



GeekBand 极客班

工具箱:

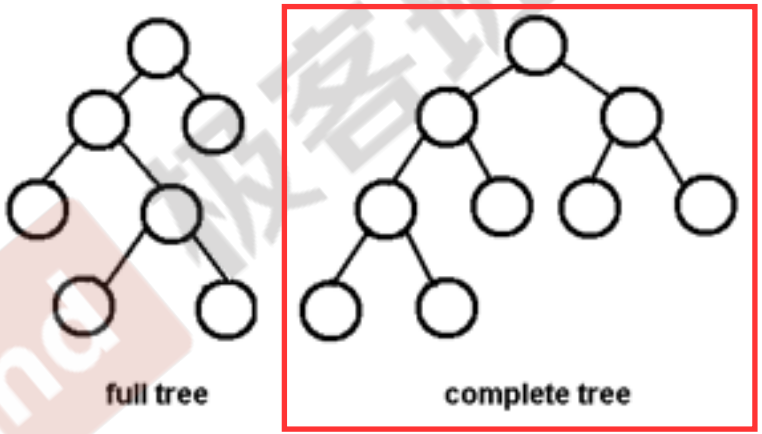
```
class TreeNode {
public:
        TreeNode *left;
        TreeNode *right;
        TreeNode *parent;
        int val;
};

class BinaryTree {
public:
    BinaryTree(int rootValue);
    ~BinaryTree();
    bool insertNodeWithValue(int value);
    bool deleteNodeWithValue(int value);
    void printTree();

private:
    TreeNode *root;
};
```

GeekBand 极客班

二叉树概念

满二叉树(full binary tree)：

完全二叉树(complete binary tree)：



层数：对一棵树而言，从根节点到某个节点的路径长度称为该节点的层数(level)，根节点为第0层，非根节点的层数是其父节点的层数加1。

高度：该树中层数最大的叶节点的层数加1，即相当于于从根节点到叶节点的最长路径加1

GeekBand 极客班

二叉树的周游

以一种特定的规律访问树中的所有节点。常见的周游方式包括：
前序周游(Pre-order traversal)：访问根结点；按前序周游左子树；按前序周游右子树。

中序周游(In-order traversal)：按中序周游左子树；访问根结点；按中序周游右子树。特别地，对于二叉搜索树而言，中序周游可以获得一个由小到大或者由大到小的有序序列。

后续周游(Post-order traversal)：按后序周游左子树；按后序周游右子树；访问根结点。

GeekBand 极客班

DFS

三种周游方式都是深度优先算法(depth-first search)

深度优先算法最自然的实现方式是通过递归实现，事实上，大部分树相关的面试问题都可以优先考虑递归。

深度优先的算法往往都可以通过使用栈数据结构将递归化为非递归实现。

GeekBand 极客班

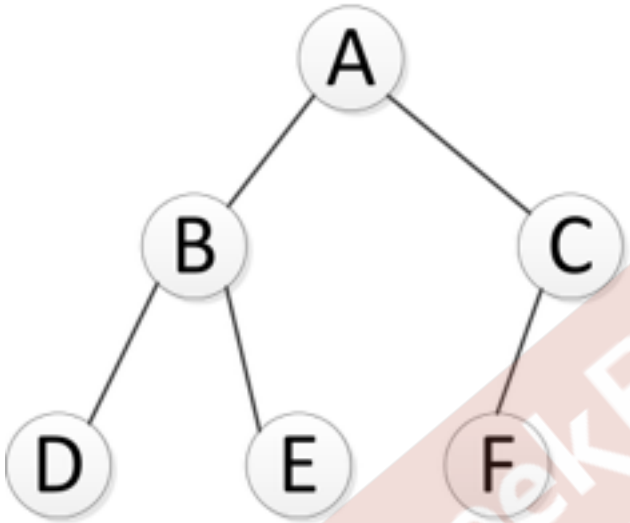
层次周游

层次周游(Level traversal)：首先访问第0层，也就是根结点所在的层；当第i层的所有结点访问完之后，再从左至右依次访问第i+1层的各个结点。

层次周游属于广度优先算法(breadth-first search)。

GeekBand 极客班

Binary Tree Traversal



pre-order: A BDE CF
in-order: DBE A FC
post-order: DEB FC A
level-order: A BC DEF

GeekBand 极客班

DFS 代码

```
void preOrderTraversal(TreeNode *root) {
    if (!root) {
        return;
    }
    visit(root);
    preOrderTraversal(root->left);
    preOrderTraversal(root->right);
}

void inOrderTraversal(TreeNode *root) {
    if (!root) {
        return;
    }
    inOrderTraversal(root->right);
    visit(root);
    inOrderTraversal(root->left);
}

void postOrderTraversal(TreeNode *root) {
    if (!root) {
        return;
    }
    postOrderTraversal(root->left);
    postOrderTraversal(root->right);
    visit(root);
}
```

GeekBand 极客班

非递归实现

```
vector<int> preorderTraversal(TreeNode *root) {
    vector<int> result;
    if (root == NULL) return result;

    stack<TreeNode *> s;
    s.push(root);
    while (!s.empty()) {
        TreeNode *node = s.top();
        s.pop();
        result.push_back(node->val);
        if (node->right != NULL) {
            s.push(node->right);
        }
        if (node->left != NULL) {
            s.push(node->left);
        }
    }

    return result;
}
```

GeekBand 极客班

BFS代码

```
void levelTraversal(TreeNode *root)
{
    queue<TreeNode *> nodeQueue;
    TreeNode *currentNode;
    if (!root) {
        return;
    }
    nodeQueue.push(root);
    while (!nodeQueue.empty()) {
        currentNode = nodeQueue.front();
        processNode(currentNode);
        if (currentNode->left) {
            nodeQueue.push(currentNode->left);
        }
        if (currentNode->right) {
            nodeQueue.push(currentNode->right);
        }
        nodeQueue.pop();
    }
}
```

GeekBand 极客班

分治算法

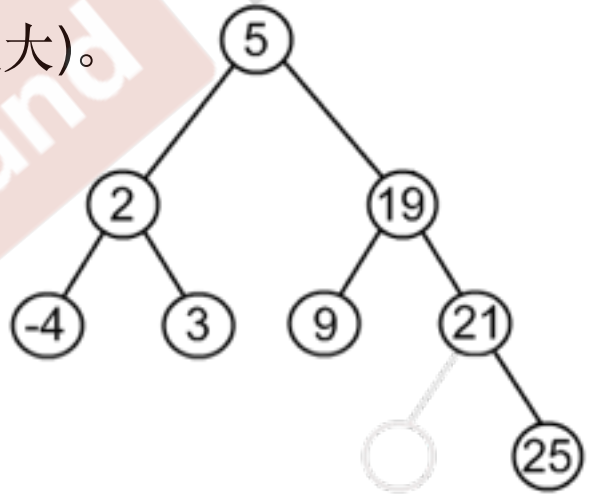
分解（Divide）：将原问题分解为若干子问题，这些子问题都是原问题规模较小的实例。
解决（Conquer）：递归地求解各子问题。如果子问题规模足够小，则直接求解。
合并（Combine）：将所有子问题的解合并为原问题的解。

- 二分搜索
- 大整数乘法
- 归并排序
- 快速排序

GeekBand 极客班

Binary Search Tree

二分查找树(Binary Search Tree, BST)是二叉树的一种特例，对于二分查找树的任意节点，该节点储存的数值一定比左子树的所有节点的值大比右子树的所有节点的值小(该节点储存的数值一定比左子树的所有节点的值小比右子树的所有节点的值大)。



GeekBand 极客班

BST特性

由于二叉树第L层至多可以储存2^L个节点，故树的高度应在logn量级，因此，二叉搜索树的搜索效率为O(logn)。

当二叉搜索树退化为一个由小到大排列的单链表（每个节点只有右孩子），其搜索效率变为O(n)。

GeekBand 极客班

Balanced Binary Tree

Determine if a binary tree is a balanced tree.

一颗二叉树是平衡的，当且仅当左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。

GeekBang 极客班

更好的实现?

一种改进方式是，可以考虑利用动态规划的思想，将TreeNode指针作为key，高度作为value，一旦发现节点已经被计算过，直接返回结果，这样，level函数对每个节点只计算一次。

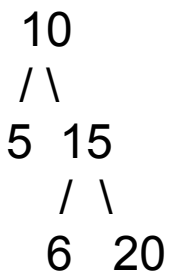
另一种更为巧妙的方法是，isBalanced返回当前节点的高度，用-1表示树不平衡。 将计算结果自底向上地传递，并且确保每个节点只被计算一次，复杂度O(n)。

GeekBand 极客班

Is Binary Search Tree

错误例子:

```
bool isValidBST(TreeNode* root) {
        if(root==NULL)
            return true;
        else if(root->left&&!root->right)
            return isValidBST(root->left)&&(root->val>root->left->val);
        else if(!root->left&&root->right)
            return isValidBST(root->right)&&(root->val<root->right->val);
        else if(root->left&&root->right)
            return isValidBST(root->left)&&isValidBST(root->right)
&&(root->val>root->left->val)&&(root->val<root->right->val);
        else
            return true;
    }
```



GeekBand 极客班

一个小技巧是：可以同时传入最小／最大值，并且将初始值设为INT_MIN，INT_MAX，这样，其左子树所有节点的值必须在 INT_MIN及根节点的值之间，其右子树所有节点的值必须在根节点的值以及INT_MAX之间。

GeekBand 极客班

Subtree

Tree1 and Tree2 are both binary trees nodes having value, determine if Tree2 is a subtree of Tree1.

GeekBand 极客班

Tree Depth

Compute the depth of a binary tree.

```
int treeDepth(TreeNode *node) {
    if (node == NULL)
        return 0;
    else
        return max(treeDepth(node->left),
treeDepth(node->right)) + 1;
}
```

GeekBand 极客班

Tree的Path问题

找出一条满足特定条件的路径 。对于这类问题，通常都是传入一个vector记录当前走过的路径(为尽可能模版化，统一记为path)，

还需要传入另一个vector引用记录所有符合条件的path(为尽可能模版化，统一记为result)。

注意， result可以用引用或指针形式，相当于一个全局变量，或者就开辟一个独立于函数的成员变量。由于path通常是vector<int> ，那么result就是vector<vector<int>>。

GeekBand 极客班

Path Sum

Get all the paths (always starts from the root) in a binary tree, whose sum would be equal to given value.

GeekBand 极客班

Path Sum2

Get all the paths (always starts from the root and ends at leaf) in a binary tree, whose sum would be equal to given value.

GeekBand 极客班

Tree 与其他数据结构转换

这类题目要求将树的结构转化成其他数据结构，例如linked list, array等，或者反之，从array等结构构成一棵树。前者通常是通过树的周游，合并局部解来得到全局解，而后者则可以利用D&C的策略，递归将数据结构的两部分分别转换成子树，再合并。

GeekBand 极客班

Binary Tree to Linked Lists

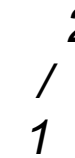
Covert a binary tree to linked lists. Each linked list is correspondent to all the nodes at the same level.

本题相当于层次周游，当周游完一层时将构成的链表加入链表集合。

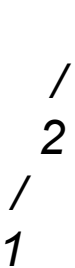
GeekBand 极客班

Sorted Array to Binary Search Tree

Convert a sorted array(increasing order) to a balanced BST.
Input: Array {1, 2, 3}
Output: A Balanced BST



Input: Array {1, 2, 3, 4}
Output: A Balanced BST



GeekBand 极客班

寻找特定节点

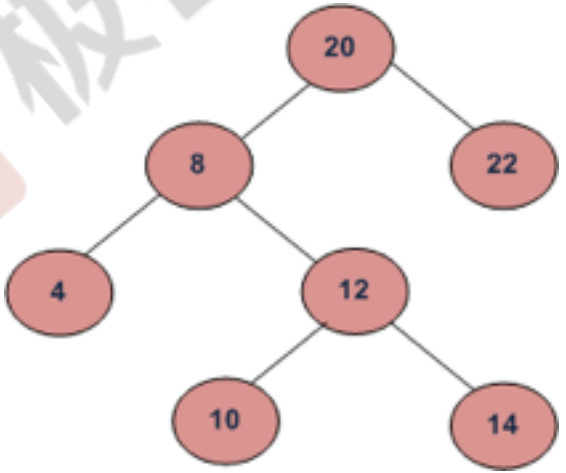
此类题目通常会传入一个当前节点，要求找到与此节点具有一定关系的特定节点：例如前驱，后继，左 / 右兄弟等。

了解一下常见特定节点的定义及性质。在存在指向父节点指针的情况下，通常可以由当前节点出发，向上倒推解决。如果节点没有父节点指针，一般需要从根节点出发向下搜索，搜索的过程就是DFS。

GeekBand 极客班

Find Next in Binary Tree

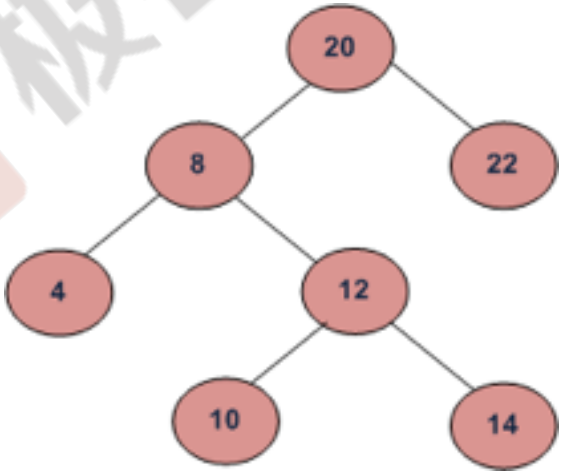
In-order traverse a binary tree with parent links, find the next node to visit given a specific node.



GeekBand 极客班

Find Next in Binary Search Tree

In-order traverse a binary search tree with parent links, find the next node to visit given a specific node.

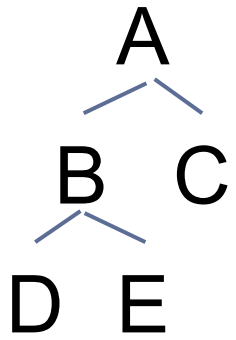


Followup: without parent link?

GeekBand 极客班

Lowest Common Ancestor(LCA)

Given a binary tree and two nodes. Find the lowest common ancestor of the two nodes in the tree.
For example, the LCA of D & E is B.



GeekBand 极客班

Right Neighbor

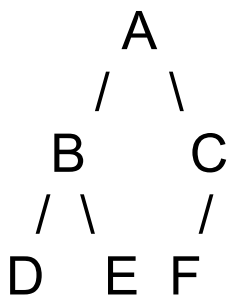
Find the immediate right neighbor of the given node, with parent links given, but without root node.

GeekBand 极客班

Rebuild Binary Tree

Pre-order + In-order

Preorder sequence: **A** B D E C F
Inorder sequence: D B E **A** F C



GeekBand 极客班

Binary Tree Level Order Traversal

- 2 Queues
- 1 Queue + Dummy Node
- 1 Queue (best)

GeekBand 极客班

Binary Tree Zigzag Level Order Traversal

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example: Given binary tree {3,9,20,#,#,15,7},

```
    3
   / \
  9  20
     /  \
    15   7
```

return its zigzag level order traversal as:

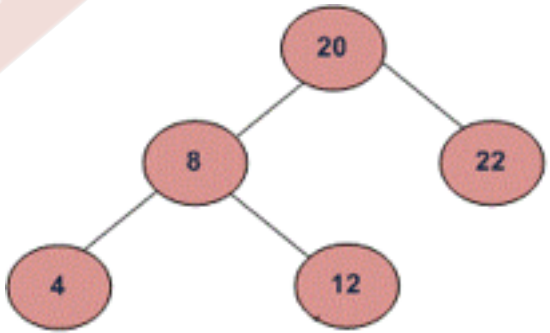
```
[
  [3],
  [20,9],
  [15,7]]
```

GeekBand 极客班

Print Range in a Binary Search Tree

Given two values k1 and k2 (where k1 < k2) and a root pointer to a Binary Search Tree. Print all the keys of tree in range k1 to k2. i.e. print all x such that k1<=x<=k2 and x is a key of given BST. Print all the keys in increasing order.

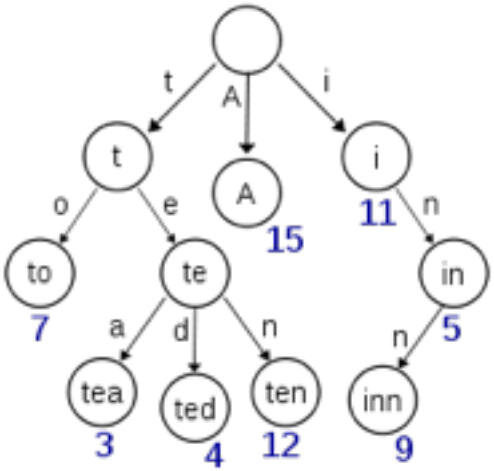
For example, if k1 = 10 and k2 = 22, then your function should print 12, 20 and 22.



GeekBand 极客班

Trie Tree

字典树(trie or prefix tree)是一个26叉树，用于在一个集合中检索一个字符串，或者字符串前缀。字典树的每个节点有一个指针数组代表其所有子树，其本质上是一个hash table，因为子树所在的位置(index)本身，就代表了节点对应的字母



GeekBand 极客班

Trie Tree定义

```
class TrieNode {
private:
    char mContent;
    bool mMarker;
    vector<TrieNode *> mChildren;
public:
    TrieNode() {
        mContent = ' '; mMarker = false;
    }
    ~TrieNode() {
    }
    friend class Trie;
};
class Trie {
public:
    Trie();
    ~Trie();
    void addWord(string s);
    bool searchWord(string s);
    void deleteWord(string s);
private:
    TrieNode *root;
};
```

GeekBand 极客班

1) void addWord(string key, int value);
添加一个key:value对。添加时从根节点出发，如果在第i层找到了字符串的第i个字母，则沿该节点方向下降一层(注意，如果下一层储存的是数据，则视为没有找到)。否则，将第i个字母作为新的兄弟插入到第i层。将key插入完成后插入value节点。

2) bool searchWord(string key, int &value);
查找某个key是否存在，并返回值。从根节点出发，在第i层寻找字符串中第i个字母是否存在。如果是，沿着该节点方向下降一层；否则，返回false。

3) void deleteWord(string key)
删除一个key:value对。删除时从底层向上删除节点，直到遇到第一个有兄弟的节点(说明该节点向上都是与其他节点共享的prefix)，删除该节点。

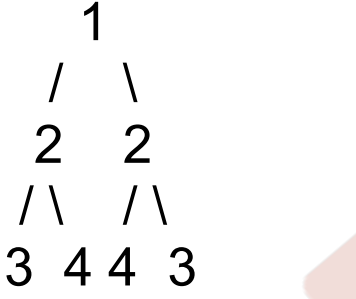
Homework

GeekBand 极客班

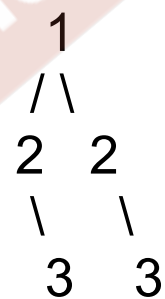
Symmetric Tree

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree is symmetric:



But the following is not:



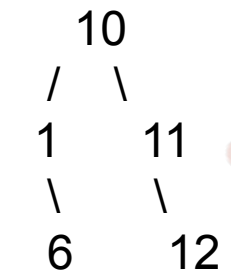
GeekBand 极客班

Binary Search Tree Iterator

Design an iterator over a binary search tree with the following rules:

- Elements are visited in ascending order (i.e. an in-order traversal)
- next() and hasNext() queries run in O(1) time in average.

Example
For the following binary search tree, in-order traversal by using iterator is [1, 6, 10, 11, 12]



参考： http://stackoverflow.com/questions/4581576/implementing-an-iterator-over-a-binary-search-tree

GeekBand 极客班

Binary Tree Level Order Traversal II

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example: Given binary tree {3,9,20,#,#,15,7},

```
    3
   / \
  9  20
     /  \
    15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]]
```

GeekBand 极客班

Sorted List to BST

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

GeekBand 极客班