0.0 template

Binary Tree DFS template
 *
 *
 * 0.0 template
 * Copyright: NineChapter
 * - Algorithm Course, Mock Interview, Interview Questions
 * - More details on: http:www.ninechapter.com/

Template 1: Traverse

```
public class Solution {
    public void traverse(TreeNode root) {
        if (root == null) {
            return;
        }
         do something with root
        traverse(root.left);
         do something with root
        traverse(root.right);
         do something with root
    }
}
```

Tempate 2: Divide & Conquer

```
public class Solution {
    public ResultType traversal(TreeNode root) {
         null or leaf
        if (root == null) {
             do something and return;
        }

         Divide
        ResultType left = traversal(root.left);
        ResultType right = traversal(root.right);

         Conquer
```

```
            ResultType result = Merge from left and right.
            return result;
        }
    }
```

1.0 preorder recursion

1.1 stack preorder use stack to memorize

---*

```
    * 1.2 divide and conquer(dac)
    *      5
    *   3    7
    * 1  4  6  9
    *
    * 1 3 4  left
    * 5      root
    * 6 7 9  right
```

---

*

```
    * 2.0
    * merge sort an array, this is not easy, using iterative method is painful.
    * merge sort has two steps, partition and merge, partition is automatically
    * finish, but how to merge
    * I start from len = 1,
    * so many details, I need to find another good implementations, anyway,
understand the thoughts of merge sort
    * is important.
    * I think merge sort > quick sort since the performance can be improved
by using multi-thread
```

*

```
    * 2.1 merge sort easy version by using recursive call
    * refer information google merge sort algorithm
    *
```

*

* 2.2 merge Sort easiest merge sort
* from professor notes

*

---

* 3.0 quick sort
* O(nlgn) O(1)
* in-place partitioning algorithm
* from professor notes

---

*

* 4. binary Tree Maximum path sum
* 记住， 一个 Node can be 整条通路
*

92 / 92 test cases passed.
Status: Accepted
Runtime: 512 ms
Submitted: 0 minutes ago

---

*

* 5.0 return all the keys in the giveRange in BST
* 5
* 3 7

* 1 4 6 9

---

*
* 6.0 implement an iterator in BST
* http:docs.oracle.com/javase/7/docs/api/java/util/Iterator.html
* only need to implement three methods for iterator interface
* boolean has Next()
* Object next();
* void remove(); optional operation
*
* 1. ascending order
* 2. next, hasNext, O(1).
* use queue
* 3. O(1) space, waiting for St.Huang to solve it.

---

*
* 6.1 from stackOverFlow
* do not need to traverse which spend O(h) time at the beginning

---

*
* 6.2 LRU Cache
* Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

1. get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

2. set(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

```
 *
```

```
*
 *
 * 7.0
 * BST template (level Order traversal)
```

```
*
 * 7.1 Zagzig traversal
```

```
*
 * 7.2
 * Clone Graph
 *
```

```
class GraphNode {
        String label;
        List<GraphNode> neighbors;
        GraphNode(String label) {
                this.label = label;
                neighbors = new ArrayList();
        }
```

```
*
 * 7.3.0
 * Word Ladder I
 *
```

*

* Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

Only one letter can be changed at a time
Each intermediate word must exist in the dictionary
For example,

Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.

Note:
Return 0 if there is no such transformation sequence.
All words have the same length.
All words contain only lowercase alphabetic characters.

---

*

* 7.3.1
* Word Ladder II BFS + DFS
Word Ladder II Total Accepted: 8944 Total Submissions: 80092 My Submissions
Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:
Only one letter can be changed at a time
Each intermediate word must exist in the dictionary
For example,
Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]

Return
[
["hit","hot","dot","dog","cog"],
["hit","hot","lot","log","cog"]
]

bug summary: Null Pointer Exception ->forget map.put(last, lastone)

---

*
* 7.4
* Surrounded Regions
For example,
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:
X X X X
X X X X
X X X X
X O X X

---

*
* 8. validate BST
* thinking from the definition of BST
* left < root < right

---

*

```
* 9. insert a node in BST, only insert at bottom
* if (root != null) {
* 1. value > root.val -> root.right
* 2. value < root.val -> root.left
* 3. value == root.val -> return false;  insert failed
* }
* root == null
* root = new TreeNode(value);
* return true;
```

```
*
 * 10.delete a node in BST
 *
 * there is a lot details when the delete is not leave node.
 * 前驱节点(left child rightmost node) the node that is closest to root
 * is useful in this case.
```

```
*
 * 11.0 Least Common Ancester(Divide And Conquer)
 * has a parent
 *
```

```
*
 * 11.1
 * if we only know root, the easy way to think is from up to bottom
 *
 * if return is null, means node1 and node2 is not in this subtree.
 *
```

```
*
 * leetcode (Divide and Conquer)
 * 12.0 construct tree from preorder, inorder (DAC);
 * Given preorder and inorder traversal of a tree, construct the binary tree.
 *
 * 12.1 construct tree from inorder, postorder(DAC);
 * Given inorder and postorder traversal of a tree, construct the binary
tree.
 *
 *  202 / 202 test cases passed.
        Status: Accepted
        Runtime: 488 ms
        Submitted: 0 minutes ago

 * tips: 碰到array index recursion, java 里尽量用 start + len 组合，可以少
出bugs
```

```
*
 * 13.0 Divide and conquer
 * this can be solved by divide the left, right two parts,
 * s1 = left1 + right1;
 * s2 = left2 + right2;
 *
 * but make sure the length should be same. and we can swicth them
 * if (left1 == left2 && right1 == right2 || left1 == right2 && left2 == right1)
 * return true
```

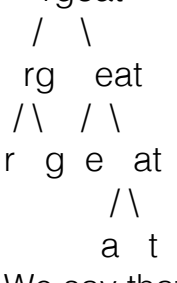
Scramble String Total Accepted: 10943 Total Submissions: 48570 My Submissions

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of s1 = "great":

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat".



We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae".

We say that "rgtae" is a scrambled string of "great".

Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

}

281 / 281 test cases passed.
Status: Accepted
Runtime: 424 ms
Submitted: 5 minutes ago

```
*
 * 14.0 Divide and Conquer + DP
 * Unique Binary Search Trees II
 *      3
 *    1   4
 * count[3] = count[0] * count[2] + count[1] * count[1] + count[2] * count[1]
 * count[n] = All Possible Of (count[left] * count[right])
 * the total problem divide into left and right subtree.
```

```
*
 * 15.0 Best Time to Buy and Sell Stock III
 * We have twice choice to buy and sell,
 * how to earn the biggest money.
 *
```

```
*
 * 16.0 Divide and Conquer
 * Flatten Binary Tree to Linked List
```
