Class 11 Recursion II

What is recursion?

Review: What is recursion (again)?

需要掌握的知识点：

1. 表象上：function calls itself
2. 实质上：Boil down a big problem to smaller ones (size n depends on size n-1, or n-2 or ...n/2)
3. **Implementation**：
    a) **1. Base case:** smallest problem to solve
    b) **2. Recursive rule.** how to make the problem smaller (if we can resolve the same problem but with a smaller size, then what is left to do for the current problem size n)

Solution:

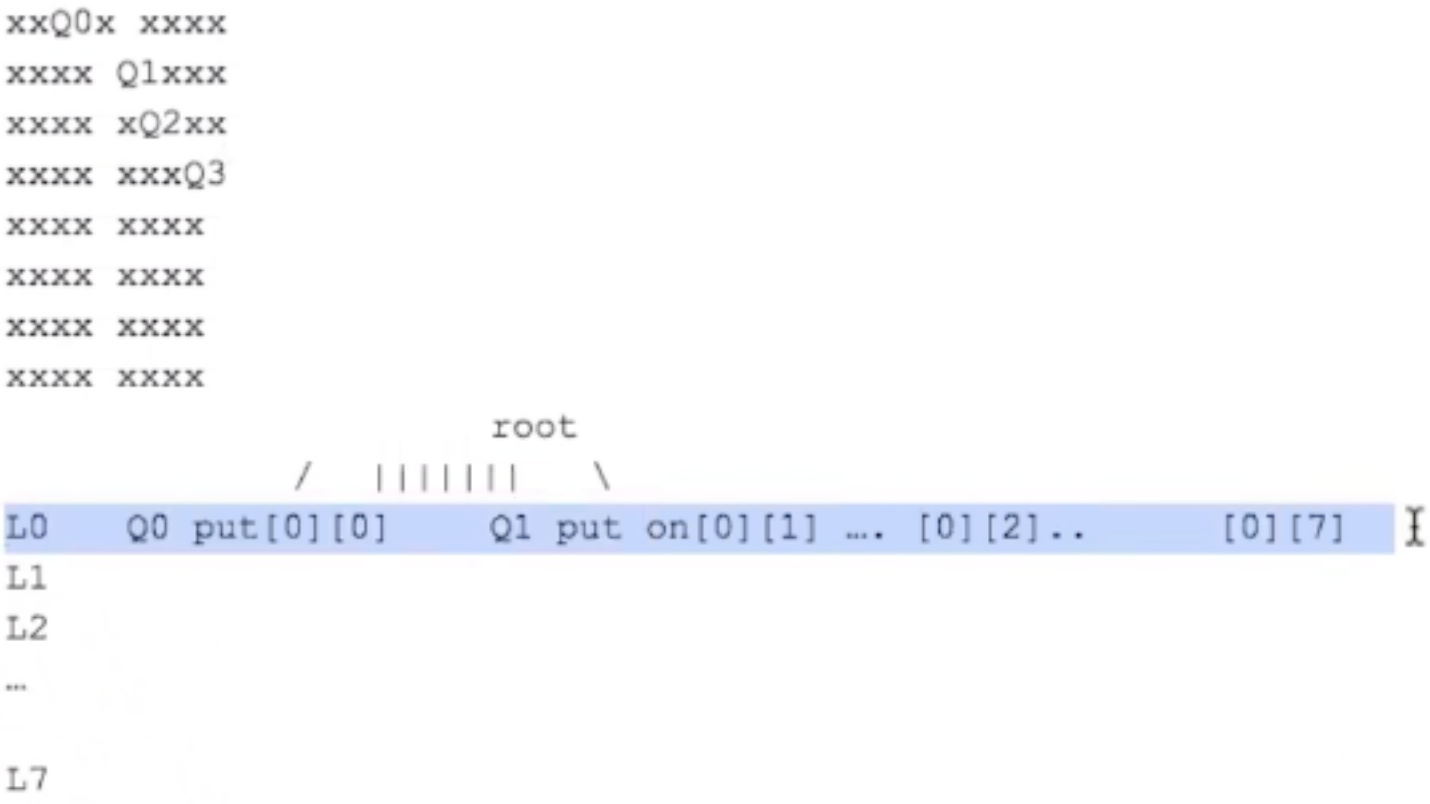
```
public double power(int a, int b) {
      if(a == 0 && b == 0) {
                  system.out.println("Error");
                  return;
      } else if(a == 0) {
         return 0;
      }
      return b < 0 ? 1 / powerHelper(a , -b) : powerHelper(a, b);
}

private double powerHelper(int a, int b) {
   if(b == 0) {
      return 1;
   }

   double half = powerHelper(a, b / 2);
   return b % 2 == 0 ? half * half : half * half *a;
}
```

Corner cases:

1. 0^0
2. denominator == 0
3. data type conversion (accuracy loss) int -- double

2. Recursion 与1D or 2D Array的结合

1. **1D array**: 二分法比较常见
    1.1. MergeSort
    1.2. QuickSort
2. **2D array**:
    2.1. 逐层(row by row)递归: 8 queen → n queen


xxQ0x xxxx
xxxx Q1xxx
xxxx xQ2xx
xxxx xxxQ3
xxxx xxxx
xxxx xxxx
xxxx xxxx
xxxx xxxx
root
/ |||||| \
L0 Q0 put[0][0] Q1 put on[0][1] …. [0][2].. [0][7]
L1
L2
…
L7

Q2.2 How to print 2D array in spiral order （NxN）

[0][0]

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Feature1 : size -= 2

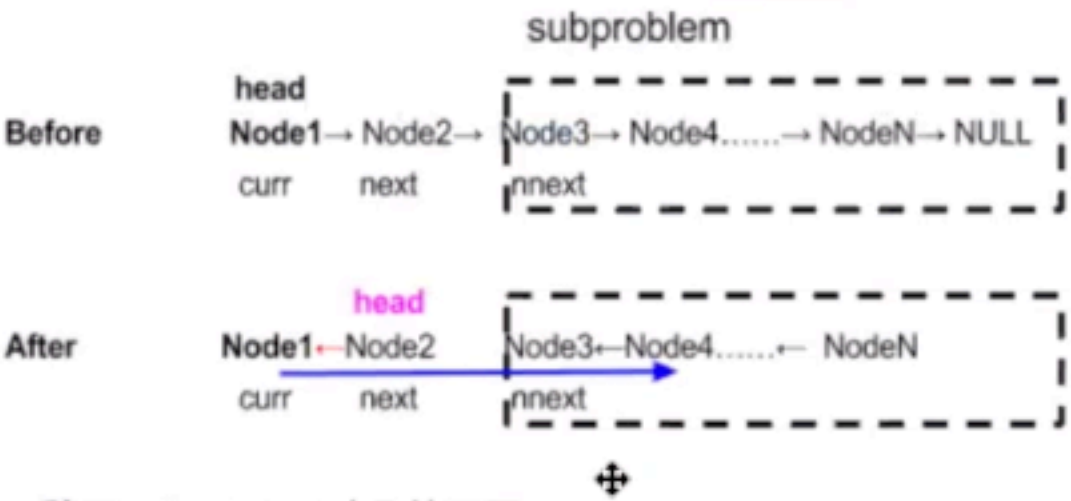
Feature2: shift +=1

3.2 Reverse a linked list (pair by pair)

Example 1-> 2 -> 3 -> 4 -> 5 → NULL

prev cur next

ouput : 2 -> 1 -> 4 -> 3 -> 5 → NULL;

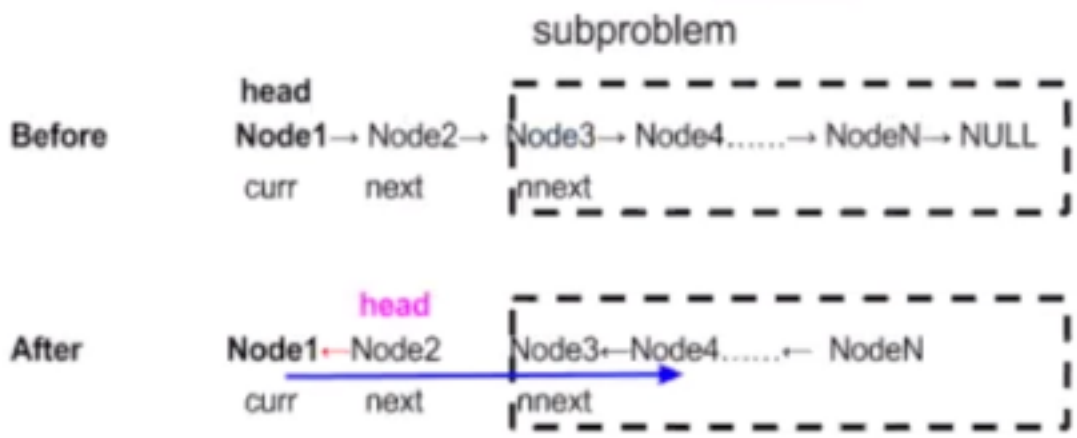


除了subproblem外几处不同？

(1) curr → next = Newhead of the subproblem; // Node 4

(2) next→ next = curr; // "next" becomes the new head;

So, if we can resolve the subproblem first, then we just need to perform (1) and (2) to solve the whole problem :)

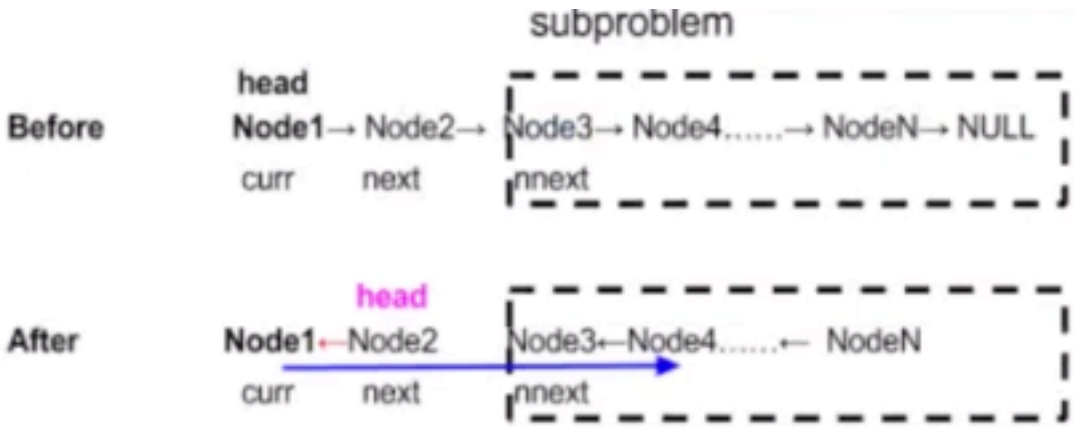


除了subproblem外几处不同？

(1) curr → next = Newhead of the subproblem; // Node 4

(2) next→ next = curr; // "next" becomes the new head;

So, if we can resolve the subproblem first, then we just need to perform (1) and (2) to solve the whole problem :)



除了subproblem外几处不同？

(1) curr → next = Newhead of the subproblem; // Node 4

(2) next→ next = curr; // "next" becomes the new head;

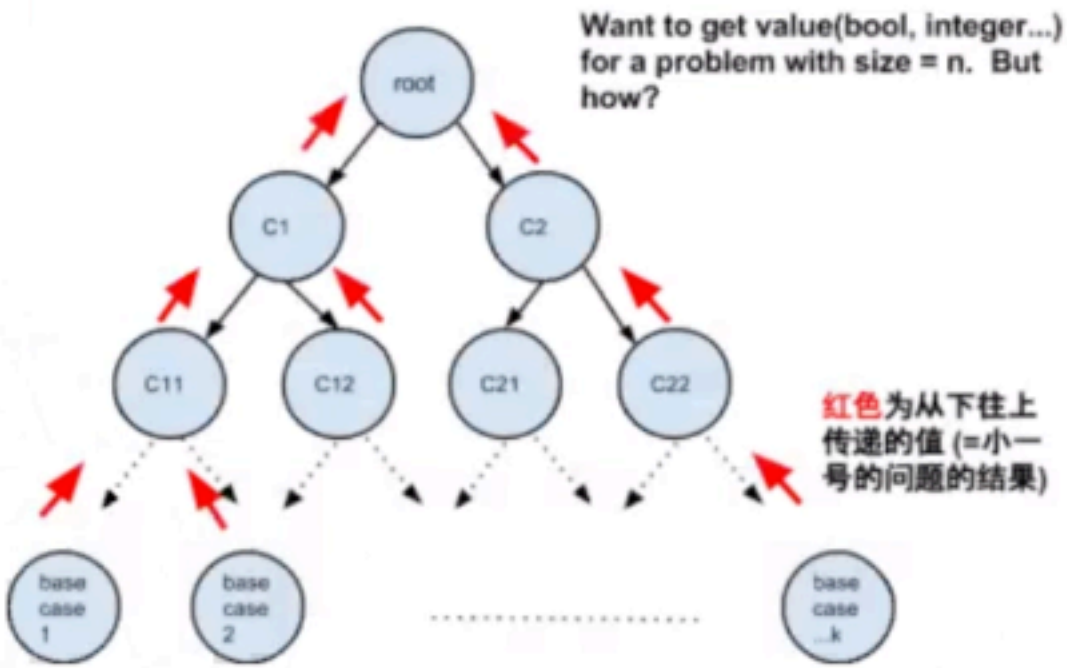
So, if we can resolve the subproblem first, then we just need to perform (1) and (2) to solve the whole problem :)

Solution:

```
public ListNode reverse (ListNode head) {
        if (head == null || head.next == null) {
                return head;
        }
        ListNode node2 = head.next;
        ListNode newHead = reverse(head.next.next);
        head.next.next = head;
        head.next = newHead;
        return node2;
```

5. Recursion 与 Tree的结合

- Binary tree 往往是最常见的，和recursion 结合最紧密的面试题目类型。
- Reasons:
  - 每层的node 具备的性质，传递的值和下一层的性质 往往一致。比较容易定义 **recursive rule**.
  - **Base case** (generally): null pointer under the leaf node
  - Example1: `int getHeight(Node root)`
  - Example2: 统计tree里边有多少个node?



Way of thinking (Tricks)

1. What do you expect from your lchild / rchild? (usually it is the return type of the recursion function)

left subtree's height **(1)**

right subtree's height **(2)**

2. What do you want to do in the current layer?

max (left, right)

3. What do you want to report to your parent? (same as Q1 == Q3)

Way of thinking (Tricks)

1. What do you expect from your lchild / rchild? (usually it is the return type of the recursion function)

left subtree's height **(1)**

right subtree's height **(2)**

2. What do you want to do in the current layer?

maxHeight = max (left, right)

3. What do you want to report to your parent? (same as Q1 == Q3)

return maxHeight **+ 1**

```
class TreeNode {
        TreeNode left;
        TreeNode right;
        int value;
        int total_left;  // how many nodes belong to its left-subtree.
}
```

Way of thinking (Tricks)

1. What do you expect from your lchild / rchild? (usually it is the return type of the recursion function)

left subtree total node **(1)**

right subtree total node **(2)**

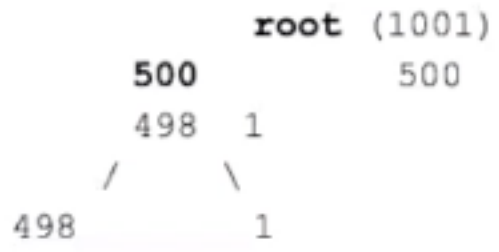
2. What do you want to do in the current layer?

fill in the value by setting Current.total_left = (1)

3. What do you want to report to your parent? (same as Q1 == Q3)

return 1 + (1) + (2)

Q5.1.3 (从下往上返值) Find the node with the **max difference** in the total number of descendents in its left subtree and right subtree



Way of thinking (Tricks)

1. What do you expect from your lchild / rchild? (usually it is the return type of the recursion function)

Total number of nodes in my left subtree **(1)**

Total number of nodes in my right subtree **(2)**

2. What do you want to do in the current layer?

2.1 calculate the diff between (1) and (2)

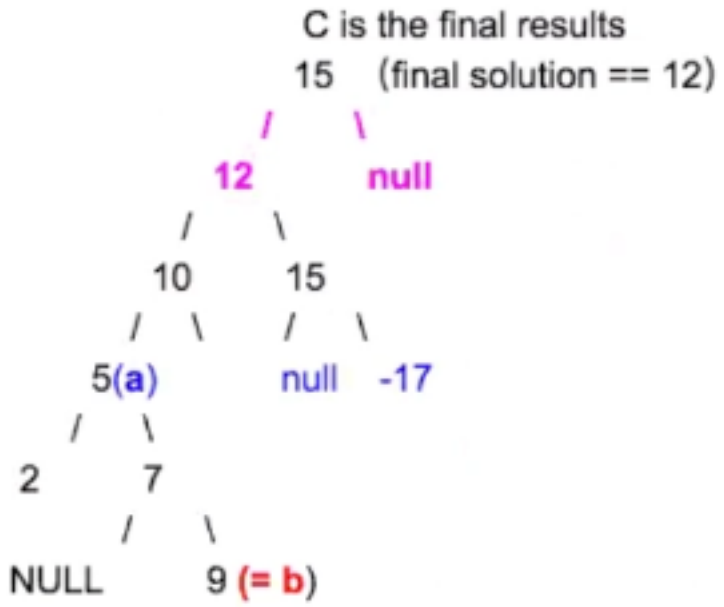
2.2 if (diff > global_diff_max) {

```
global_diff_max = diff;
solu_node = current_node;
}
```

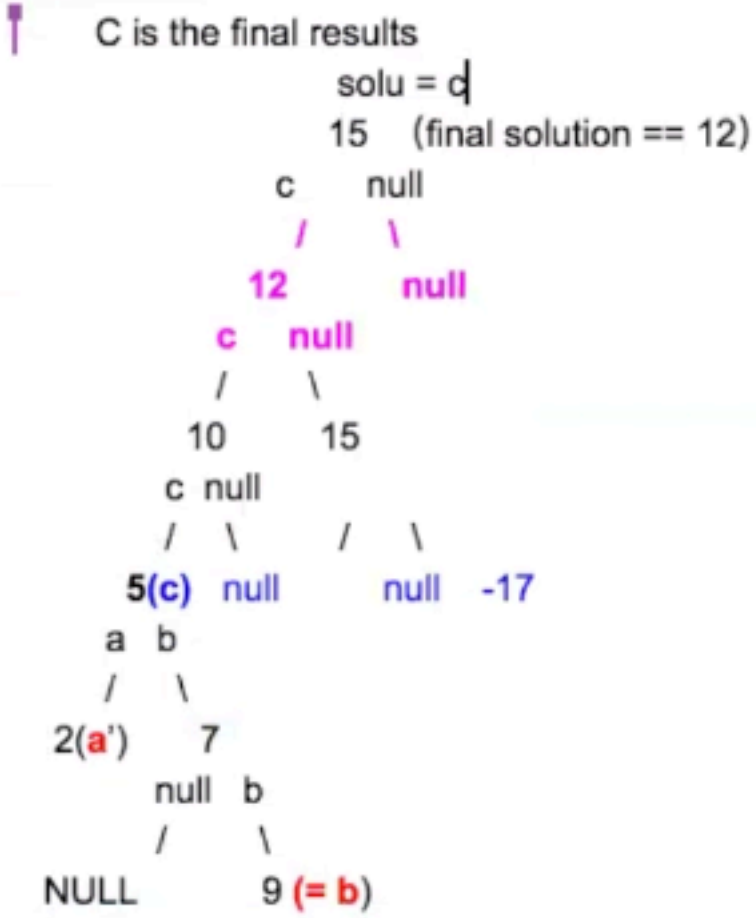
3. What do you want to report to your parent? (same as Q1 == Q3)

return 1 + (1) + (2)

Q5.1.4 (从下往上返值，最经典例题) Lowest Common Ancestor (LCA)



C is the final results



(1) a and b 非直接隶属

(1) **a and b 非直接隶属**

(a) Case1: if both sides are null, return null

(b) Case2: one side return not null, and the other side is null, return NON null side

(c) Case3: if both sides are NOT null, return c

Assumption:
a or b might NOT be in the tree

1) Case1 : return c
2) Case2: return null;
3) Case2: **return a** or b

```
void main () {
        // we construct a new tree
        provide you   two node pointers
        TreeNode solu = LCA (root, a, b);
        if (solu != a && solu != b) {
                cout << "solution is found which is c" ;
        } else if  (solu == null) {
                cout<<  "neither a nor b is in the tree";
```

```
        } else {
                if (solu == a) {
                        if (b == LC{A(a, b, null)) {
                                cout << "a is the final solution"
                        }  else {
                                cout << "b is not in the tree, so no solution";
                        }
                } else {
                        ????
                }


        }

}

}
```

==========

Given a very big tree with 1 Trillion nodes, and 1024 machines, how can we get the solution in parallel??

root

L0

L1

L2

L8

L9 (1024)

M9_1 M9_2 M9_3 M9_4 … M9_1024

Step1: do analysis (pre-processingh) RUN BFS1, to determine whether a or b is in the top 10 levels.

Step1: do analysis (pre-processing) run BFS1, to determine whether a or b is in the top 10 levels.

Case1.1 if both a and b are in top 10 levels. run LCA (root, a, b, 10) among top 10 levels., that is it!

Case1.2 if only one a or b is in top 10 levels. Let's say a is found in top 10 levels.

Use 1024 machinese to find b, let's say **M9_i** found b.

return **LCA** (root, a, node_9_i, 10)

Case1.3 if neither a nor b is in top 10 levels.

Case1.3 if neither a nor b is in top 10 levels.

run **LCA(M_9_i, a, b)**

Case1.3.1. say one machine M_9_i node return c, so c is the final result

Case1.3.2 say one machine M_9_i node return a, and the machinese M_9_j node return b;, then return LCA(root, node_9_i, node_9_j , 10);

Case1.3.3 say one machine M_9_i node return a, all other machinereturn null. check the subtree rooted at a, to find whether b is in the subtree under a. !

Follow up 2: (Dexiang)

Given a binary tree, and a list of treeNode, (k nodes) return lowest common ancestor

Method1: (Yajing Cai) binary reduction

1

2 12

3

4 34 14 k nodes

5

6 56 18

7

8 78 58

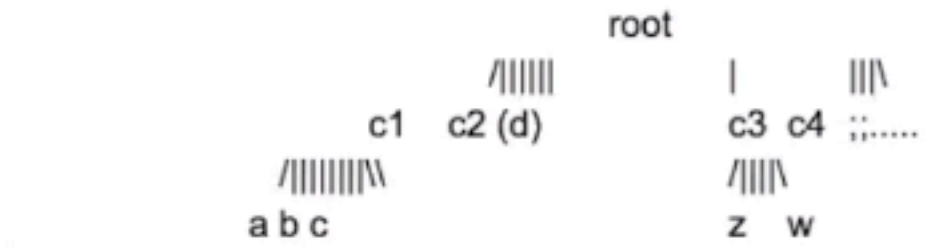
log(k) column

(½)k LCA (¼ k) LCA (1/8k) LCA

(½ + ¼ + ⅛ + ….)k LCA = 1 * K * LCA = O(kn)

```
public TreeNode LCA (TreeNode root, set<TreeNode> input) {
      if (root == null || input.contains(root)) {
            return root;
      }
      TreeNode left = LCA(root.left, input);       // step 1
      TreeNode right = LCA(root.right, input);
```

```
      if(left != null & right != null) {
            return root;                           // step 2 & 3
      }
      return left == null ?  right : left;
}
```




```
public TreeNode LCA (TreeNode root, set<TreeNode> input) {
      if (root == null || input.contains(root)) {
            return root;
      }
      int count = 0;
      TreeNode solu = null;
      for(TreeNode cur: root.child) {
            TreeNode temp = LCA(cur, input);
            if (temp != null){
                  count++;
                  if (count == 2) return root;
                  solu = temp;
            }
      }
      return solu;
}
```