Must write within 1

Remember:

```
        String --->  str.length();
        Array---> arr.length;
        List--->int len=list.size();

        String.valueOf(char[] ch);



        Arrays.sort(array[])  //对数组进行快排
    System.out.print(Arrays.toString(array));//输出快排后的数组  ---> [1,2,7,9,11]

        Collections.sort(List<xxxx>);

        Collections.reverse(list);

        //static int  binarySearch(List list, Object key) 使用二分查找法查找指定元素在指定列表的索引位置
        int index = Collections.binarySearch(list, 4);


    //static void swap(List list, int i, int j) :将指定列表中的两个索引进行位置互换
    Collections.swap(list, 0, 1);



        int[] int_n=new int[10];
        // clone the array
        int_n.clone(xxx[]);



     // copying array org to copy
     copyOf(int[] original, int newLength)
    int[] copy = Arrays.copyOf(org, 5);
```

Arrays:

java.util.Arrays类是数组的工具类，一般数组常用的方法包括

二分查找：public static int binarySearch(array[],int key)，返回key的下标index

扩容缩容：public static int[] copyOf(array[]，newLength)，返回新数组

取部分：public static int[] copyOfRange(array[],fromindex,toindex) ，注意[from,to)是开区间，返回新数组

数组从小到大快速排序：public static void Arrays.sort(array[]),

整体赋值：public static void fill(array[],value),

输出为字符串：public static String toString(array[])，返回字符串

检查是否相等：public static boolean equals(array1[],array2[])，返回布尔值

Collections

```
 //对集合进行排序
    Collections.sort(list);

    //寻找最大和最小值

    int max = Collections.max(list);
    int min = Collections.min(list);

      List<String> list2 = Arrays.asList("Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday".split(","));
    System.out.println(list2);


//查找子串在集合中首次出现的位置
    List<String> subList = Arrays.asList("Friday,Saturday".split(","));
    int index3 = Collections.indexOfSubList(list2, subList);
    System.out.println(index3);

   //反转集合中的元素的顺序
    Collections.reverse(list2);
    System.out.println(list2);

 //交换集合中指定元素的位置
        Collections.swap(list2, 0, 3);
        System.out.println(list2);

 //为集合生成一个Enumeration
    List<String> list5 = Arrays.asList("I love my country!".split(" "));
    System.out.println(list5);
    Enumeration<String> e = Collections.enumeration(list5);
    while (e.hasMoreElements()) {
        System.out.println(e.nextElement());
    }


原文链接：https://blog.csdn.net/wangshuang1631/article/details/53200764
```

Clone the Undirected graph.

http://n00tc0d3r.blogspot.sg/2013/09/clone-graph.html

Each node in the graph contains a label and a list of its neighbors.

```
class UndirectedGraphNode {
      int label;
      ArrayList<UndirectedGraphNode> neighbors;
      UndirectedGraphNode(int x) { label = x; neighbors = new ArrayList(); }
  };
```

Implementation with DFS

```
   private UndirectedGraphNode cloneDFS(UndirectedGraphNode root, HashMap<UndirectedGraphNode, UndirectedGraphNode> visited) {
     if (root == null) return root;
     UndirectedGraphNode node = new UndirectedGraphNode(root.label);
     visited.put(root, node);

     // DFS
     for (UndirectedGraphNode nb : root.neighbors) {
       if (visited.containsKey(nb)) {
         node.neighbors.add(visited.get(nb));
       } else {
         node.neighbors.add(cloneDFS(nb, visited));
       }
     }

     return node;
   }

   public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
     return cloneDFS(node, new HashMap<UndirectedGraphNode, UndirectedGraphNode>());
   }
```

Implementation with BFS

```
  public UndirectedGraphNode cloneBFS(UndirectedGraphNode root) {
    if (root == null) return root;

    ArrayDeque<UndirectedGraphNode> que = new ArrayDeque<UndirectedGraphNode>();
    que.addLast(root);

    HashMap<UndirectedGraphNode, UndirectedGraphNode> visited = new HashMap<UndirectedGraphNode, UndirectedGraphNode>();
    UndirectedGraphNode rootCopy = new UndirectedGraphNode(root.label);
    visited.put(root, rootCopy);

    // BFS
   while (!que.isEmpty()) {
       root = que.removeFirst();
       UndirectedGraphNode node = visited.get(root);

       for (UndirectedGraphNode nb : root.neighbors) {
           if (visited.containsKey(visited.get(nb))) {
               node.neighbors.add(visited.get(nb));
           } else {
               UndirectedGraphNode n = new UndirectedGraphNode(nb.label);
               node.neighbors.add(n);
               visited.put(nb, n);
               que.addLast(nb);
           }
       }
    }
     return rootCopy;
   }
```

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.--KMP

```
    public int strStr3(String haystack, String needle) {
        // Input validation.
        if (haystack == null || needle == null) return -1;
        if (haystack.length() < needle.length()) return -1;
//          if (needle.length() == 0) return haystack;
            if(needle.equals("")&&haystack.equals("")) return 0;
            if(needle.length()==0) return -1;

//          if(needle.length()==0) return 0; // according to leetcode: Input: "" "" Expect: 0

        // KMP.
        int firstMatchedIndex = KMP(haystack, needle);

        // Return result.
        if (firstMatchedIndex < 0) return -1;
//          else return haystack.substring(firstMatchedIndex, haystack.length());
            else return firstMatchedIndex;
        }

    public int[] getNextArr(String pat) {
        int pat_len = pat.length();

        int[] next = new int[pat_len];
        next[0] = -1;

        int prefix_index = -1;
        int suffix_index = 0;

        while (suffix_index < pat_len) {
            if (prefix_index == -1 ||
                pat.charAt(prefix_index) == pat.charAt(suffix_index)) {

                int numMatched_prefix_and_suffix = prefix_index + 1;

                if (suffix_index + 1 >= pat_len) break;
                next[suffix_index + 1] = numMatched_prefix_and_suffix;

                prefix_index ++;
                suffix_index ++;

            } else {
                // Let next array guide use how to reset the prefix_index.
                prefix_index = next[prefix_index];

            }
        }

        return next;
```

```
    }

    public int KMP(String str, String pat) {
        int str_len = str.length();
        int pat_len = pat.length();

        int[] next = getNextArr(pat);

        int str_start = 0;
        int pat_start = 0;

        while (str_start < str_len && pat_start < pat_len) {
            // Since if the pat_start = next[0], the pat_start == -1,
            // so we need check whether pat_start == -1,
            // if that we just keep move forward.
            if (pat_start == -1 || str.charAt(str_start) == pat.charAt(pat_start)) {

                str_start ++;
                pat_start ++;

            } else {
                // Let next array guide us how to reset the pat_start.
                pat_start = next[pat_start];

            }
        }

        if (pat_start >= pat_len) {
            // The current str_start is point to,
            // the end of the matched part in the str,
            // and we know the matched length is pat_len,
            // so we use str_start - pat_len to get,
            // the matched start point in str.
            return str_start - pat_len;
        } else return -1;
    }
```

Format the code to be readable:

```
   public int findSubStrBeginIndex(String str, String subStr) {
       // Input validation.
       if (str == null || subStr == null) return -1;
       if (str.length() < subStr.length()) return -1;

       if(subStr.equals("")&&str.equals("")) return 0;
       if(subStr.length()==0) return -1;

       // KMP.
       int firstMatchedIndex = KMP(str, subStr);

       // Return result.
       if (firstMatchedIndex < 0) return -1;
       else return firstMatchedIndex;
   }

public int[] getNextArr(String subStr) {
   int subStr_len = subStr.length();

   int[] next = new int[subStr_len];
   next[0] = -1;

   int prefix_index = -1;
   int suffix_index = 0;

   while (suffix_index < subStr_len) {
       if (prefix_index == -1 ||
               subStr.charAt(prefix_index) == subStr.charAt(suffix_index)) {

           int numMatched_prefix_and_suffix = prefix_index + 1;

           if (suffix_index + 1 >= subStr_len) break;
           next[suffix_index + 1] = numMatched_prefix_and_suffix;

           prefix_index ++;
           suffix_index ++;

       } else {
           // Let next array guide use how to reset the prefix_index.
           prefix_index = next[prefix_index];

           }
       }

       return next;
}

  public int KMP(String str, String subStr) {
        int str_len = str.length();
        int subStr_len = subStr.length();

        int[] next = getNextArr(subStr);

        int str_start = 0;
        int subStr_start = 0;

        while (str_start < str_len && subStr_start < subStr_len) {
            if (subStr_start == -1 || str.charAt(str_start) == subStr.charAt(subStr_start)) {

                str_start ++;
                subStr_start ++;

            } else {
                subStr_start = next[subStr_start];


            }
        }

        if (subStr_start >= subStr_len) {
            return str_start - subStr_len;
        } else return -1;
    }
```

Given a sorted (in increasing order) array with unique integer elements, write an algorithm to create a binary search tree with minimal height.

```
    public void convert(int[] nums,int left,int right){

        if(left>right){
        return null;
        }

        int middle=low+((right-left)>>1);

        TreeNode root=new TreeNode(nums[middle]);

        root.left=convert(nums,left,middle-1);
        root.right=convert(nums,middle+1,right);

        return root;
    }
```

Input n,m Pick up some numbers from 1,2,3....n, to fulfill the sum of them is equal to m. --can repeat pick up numbers

```
    public void runPermutation(int[] a) {

        if(null == a || a.length == 0)
            return;

        int[] b = new int[a.length];//辅助空间，保存待输出排列数
        getAllPermutation(a, b, 0);
    }

    public void getAllPermutation(int[] a, int[] b, int index) {

        if(index == a.length){
            for(int i = 0; i < index; i++){
                System.out.print(b[i] + " ");
            }
            System.out.println();
            return;
        }

        for(int i = 0; i < a.length; i++){

            b[index] = a[i];
            getAllPermutation(a, b, index+1);
        }

    }

    public static void main(String[] args){

        Solution3 robot = new Solution3();

        int[] a = {1,2,3};
        robot.runPermutation(a);

    }
```

Input m and an input Array, pick up some numbers from specfic array, to fulfill the sum of them is equal to m. (can repeat)

```
    public static void main(String[] args){
        Solution3 robot = new Solution3();
        int[] a = {1,2,3,6,7,3,4,10};
        robot.findSum(a,10);
    }

int[] nums;
    public void findSum(int[] nums,int sum){

        // Arrays.sort(nums);     // no need sort
        this.nums=nums;
        combine(sum);
    }



        public void combine(int m) {

            if(m < 1 )
```

```
            return;

        ArrayList<Integer> arr = new ArrayList<Integer>();
        getCombination(m, arr);
    }
    public void getCombination(int m, ArrayList<Integer> arr) {

        if (m == 0 && arr.size() >= 1) {
            for (int i = 0; i < arr.size(); i++) {

                System.out.print(arr.get(i) + " ");
            }
            System.out.println();
            return;
        }

        if(m<0) return;

        for (int i = 0; i <nums.length; i++) {
            if (!arr.isEmpty() && nums[i] < arr.get(arr.size() - 1))//使集合内元素递增，防止重复
                continue;

            arr.add(nums[i]);
            getCombination(m - nums[i], arr);
            if(!arr.isEmpty())
            arr.remove(arr.size()-1);
        }
    }
```

```
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 2
1 1 1 1 1 1 1 3
1 1 1 1 1 1 1 3
1 1 1 1 1 1 2 2
1 1 1 1 1 1 4
1 1 1 1 1 2 3
1 1 1 1 1 2 3
1 1 1 1 2 2 2
1 1 1 1 2 4
1 1 1 1 3 3
1 1 1 1 3 3
1 1 1 1 3 3
1 1 1 1 3 3
1 1 1 1 6
1 1 1 2 2 3
1 1 1 2 2 3
1 1 1 3 4
1 1 1 3 4
1 1 1 7
1 1 2 2 2 2
1 1 2 2 4
1 1 2 3 3
1 1 2 3 3
1 1 2 3 3
1 1 2 3 3
1 1 2 6
1 1 4 4
1 2 2 2 3
1 2 2 2 3
1 2 3 4
1 2 3 4
1 2 7
1 3 3 3
1 3 3 3
1 3 3 3
1 3 3 3
1 3 6
1 3 3 3
1 3 3 3
1 3 3 3
1 3 3 3
1 3 6
2 2 2 2 2
2 2 2 4
2 2 3 3
2 2 3 3
2 2 3 3
2 2 3 3
2 2 6
```

```
2 4 4
3 3 4
3 3 4
3 7
3 3 4
3 3 4
3 7
4 6
10
```

Input n,m Pick up some numbers from 1,2,3....n, to fulfill the sum of them is equal to m. (0/1 bag)

```
    LinkedList<Integer> list=new LinkedList<Integer>();


    public void find_factor(int sum,int n){

        if(n<=0||sum<=0) return;  //递归退出条件

        if(sum==n){

            //reverse the list:
            Collections.reverse(list);
//          for(int i=0;i<list.size();i++){
//              int tmp=list1.get(i);
//              list1.set(i, list1.get(list1.size()-1-i));
//              list1.set(list1.size()-1-i, tmp);
//          }
//
            for(int tmp:list)
                System.out.print(tmp+"+");

            System.out.println(n);
        }

        list.push(n);  // save the current try
        find_factor(sum-n,n-1);  // 将改值的Wi 放入背包，减少当前容量，并在n-1范围内再试，看能否到满足条件临界。
        list.pop();// 还原状态，不将当前值放入背包
        find_factor(sum,n-1); // 尝试 不将当前值放入背包，在n-1范围内再试。

    }
```

====> another way to write the code:

```
LinkedList<Integer> list=new LinkedList<Integer>();

public void find_factor(int sum,int n){
    if(sum<=0 ||n<=0) return;

    if(sum==n){
        for(int i=list.size()-1;i>0;i--)
        System.out.print(list.get(i)+" ");
        System.out.println();
    }
    list.push(n);
    find_factor(sum-n,n-1);
    list.pop();
    find_factor(sum,n-1);
}
```

Output:

```
s.find_factor(10,20);

10
9+1
8+2
7+3
7+2+1
6+4
6+3+1
5+4+1
4+3+2
4+3+2+1
```

Binary operation / Bit operations --- The sum of two binary numbers

```
public static String addBinary4(String a, String b){
    if (a == null || a.length() == 0)
        return b;
    if (b == null || b.length() == 0)
        return a;

    int currA = a.length() - 1;
    int currB = b.length() - 1;
    int flag = 0;
    StringBuilder sb = new StringBuilder();

    while (currA >= 0 || currB >= 0) {

        int va = 0;
        int vb = 0;

        if (currA >= 0) {
            va = a.charAt(currA) == '0' ? 0 : 1;
            currA--;
        }

        if (currB >= 0) {
            vb = b.charAt(currB) == '0' ? 0 : 1;
            currB--;
        }

        int sum = va + vb + flag;

        if (sum >= 2) {
            sb.append(sum - 2);
            flag = 1;
        } else {
            sb.append(sum);
            flag=0;
        }

    }

    if (flag == 1) {
        sb.append("1");
    }

     return sb.reverse().toString();
}
```

PreOrder Traversal

Without using recursion:

```
   public List<Integer> preorderTraversal3(TreeNode root) {
        Stack<TreeNode> s=new Stack<TreeNode>();
        s.push(root);
        ArrayList<Integer> arr=new ArrayList<Integer>();

        while(!s.isEmpty()){
         TreeNode node=s.pop();
          if(node!=null){
//        System.out.print(node.val+" ");
             arr.add(node.val);
          //Last in first out.
            if(node.right!=null){
                s.push(node.right);
            }


            if(node.left!=null) {
                s.push(node.left);
            }
        }

        }

        return arr;

    }
```

Using recustion:

```
//By using recursion:

public List<Integer> preorderTraversal(TreeNode root) {

    if(root!=null){

        System.out.println(root.val);

        if(root.left!=null) preorderTraversal(root.left);

        if(root.right!=null) preorderTraversal(root.right);


    }


return null;
}
```

InOrder Traversal

Without using recursion:

```
public List<Integer> inorderTraversal(TreeNode root) {

    Stack<TreeNode> s = new Stack<TreeNode>();
    List<Integer> result = new ArrayList<Integer>();
    HashMap<TreeNode, Boolean> vistedMap = new HashMap<TreeNode, Boolean>();

    s.push(root);
    while (!s.isEmpty()) {
        TreeNode node = s.pop();

        if (node != null) {

            if (vistedMap.get(node) == null) {
                if (node.right != null) {
                    s.push(node.right);
                }

                if (node.left != null || node.right != null)
                    s.push(node);

                if (node.left == null && node.right == null) {
                    result.add(node.val);
                }

                if (node.left != null) {
                    s.push(node.left);
                }

                // marke it as visted.
                if (vistedMap.get(node) == null) {
                    vistedMap.put(node, true);
                }

            } else {
                result.add(node.val);
            }

        }

    return result;
}
```

Using recursion:

```
public List<Integer> inOrderTraversalRecursion(TreeNode root){

    if(root!=null){

    if(root.left!=null) inOrderTraversalRecursion(root.left);

    list.add(root.val);

    if(root.right!=null) inOrderTraversalRecursion(root.right);

    }

    return list;
}
```

PostOrder Traversal

Using recursion:

```
public List<Integer> postOrderTraversalRecursion(TreeNode root){
    if(root.left!=null) postOrderTraversalRecursion(root.left);

    if(root.right!=null) postOrderTraversalRecursion(root.right);

    list.add(root.val);

    return list;
}
```

LevelOrder Traversal

```
public static List<List<Integer>> levelOrder(TreeNode root) {

    if(root==null) return new ArrayList<List<Integer>>();

    List<List<Integer>> result=new ArrayList<List<Integer>>();

    LinkedList<TreeNode> que=new LinkedList<TreeNode>();

    List<Integer> list=new ArrayList<Integer>();

    que.add(root);
    que.add(null);

    while(!que.isEmpty()){

        TreeNode firstNode=que.removeFirst();


        if(firstNode==null){
            result.add(list);
            //reset the list ArrayList.
            list=new ArrayList<Integer>();
            if(!que.isEmpty()) que.addLast(null);

        }else{

            list.add(firstNode.val);
            if(firstNode.left!=null) que.add(firstNode.left);
            if(firstNode.right!=null) que.add(firstNode.right);

        }

    }

    return result;
}
```

Deepth of binary tree Traversal

Linked List Traversal

Fibonacci Number

```
    public long find_fib2(int n){

          long v3=0;     //using long can reach:91 7540113804746346429

          if(n==0||n==1) return 1;

          long v1=1;
          long v2=1;

          for(int i=1;i<n;i++){
            v3=v1+v2;
            v1=v2;
            v2=v3;
          }

          return v3;

        }
```

```
    public int find_fib(int n){

      int v3=0;   //using int only can reach: 45 1836311903 after that output becomes negative number

      if(n==0||n==1) return 1;

      int v1=1;
      int v2=1;

      for(int i=1;i<n;i++){
        v3=v1+v2;
        v1=v2;
        v2=v3;
      }

      return v3;

    }


    //using recursion:
    public int find_fib0(int n){  // it is quite slow, when it show 43 701408733

        if(n==0||n==1) return 1;

        return find_fib0(n-1)+find_fib0(n-2);

        }
```

BFS

DFS

Judge whether has cycle

```
public static boolean hasCycle(ListNode head){
    ListNode first=head,second=head;

    if(head==null) return false;  // when head==null
    if(head.next==null) return false; // when only contain head node.

    // when contain more than one node
    int j=0;
    while(first.next!=null){
        first=first.next;
        j++;
        if(j==2){
            j=0;
            second=second.next;
        }
        if(first==second) return true;

    }

    return false;
}
```

find the insection node of two single lists

find the beginning node of cycle (Linked List)

```
public static ListNode detectCycle5(ListNode head) {

    ListNode first = head, second = head;

    if (head == null)
        return null; // when head==null
    if (head.next == null)
        return null; // when only contain head node.
    if(head.next.next==head) return head;

    boolean adjustFlag = false;
    // when contain more than one node

    int meetTime=0;

    while (first!=null&&first.next != null) {

        if(!adjustFlag){
        first = first.next.next;
        second=second.next;
        }else{
            first=first.next;
            second=second.next;
        }
        if(first==second&&first!=null&&meetTime!=2){
            System.out.println("meet:"+first.val);
            meetTime++;
            if(meetTime==2) return second;
            adjustFlag=true;
            second=head;
        }


    }

    return null;
}
```

LRU Cache

```
    HashMap<Integer, Node> map = new HashMap<Integer, Node>();


    int capacity = 0;

    Node head = null;
    Node rear = null;

    public LRUCache(int capacity) {
        this.capacity = capacity;
    }

    public int get(int key) {

        if(map.get(key)!=null) {
            remove(map.get(key));
            setHead(map.get(key));
            return map.get(key).value;
        }

        return -1;
    }

    public void set(int key, int value) {

           if(map.get(key)!=null){
             Node tmpNode=map.get(key);
             tmpNode.value=value;

             /*
              *  set it to the head
              */
             remove(tmpNode);
             setHead(tmpNode);
           }else{
```

```
            Node newNode=new Node(key,value);

            //add this new key and value into the HashMap
            map.put(key, newNode);

            if(map.size()<=capacity){
                setHead(newNode);
            }else{
                removeRear();
                setHead(newNode);
            }

        }
    }

    public void setHead(Node node){

        /*
         * 1. remove the element from LinkedList
         *
         * Cannot put remove element from LinkedList here:
         * This method will be called by two different cases:
         * (1) add totally new node into the list
         * (2) adjust the old node's position in the list.
         *
         * So we need to use a separated method to handle this.
         *
         *
         */

        //if this node is  head, no need to do the "set head operation"
        if(node==head) return;
    //
    //
    //      if(node.prev!=null) node.prev.next=node.next;
    //
    //      if(node.next!=null) node.next.prev=node.prev;



        /*
         * 2. move it to the head.
         *   (1) head==null
         *   (2) head!=null
         */
        if(head==null) {
            head=node;
            rear=head;
        }else{
            node.prev=null;
            head.prev=node;
            node.next=head;
            head=node;
        }




    }


    public void remove(Node node){
        /*
         * For removing node, there are several cases:
         *
         * remove from the head
         * remove from the rear
         *
         * remove from middle
         *
         *
         */

        // Consider two directions:
        // from the next direction
        if(node.next!=null){



            if(node.prev!=null){

            }
```

```
            //if node.preve==null, that means it is head, adjust the head pointer
            else{
                head=node.next;
            }

            node.next.prev=node.prev;
//

            //if node.next==null, that means it is rear. adjust the rear pointer.
        }else{

            rear=node.prev;

            if(node.prev!=null){
            node.prev.next=null;
            }
            //node.prev==null, it is head.
            else{
                head=null;
                rear=null;
            }

      }


    // Consider another direction:
    // from the another direction:


      if(node.prev!=null){

          if(node.next!=null){

          }
          //if node.next==null, it means it is rear. need to adjust the rear pointer
          else{
              rear=node.prev;
          }

          node.prev.next=node.next;

      }
      //if node.prev==null, that means it is head, need to adjust the head pointer
      else{

          head=node.next;

          if(node.next!=null){
              head.prev=null;

          //node.next==null, it is rear.
          }else{
              head=null;
              rear=null;
          }
      }



}


   public void removeRear(){

       /*
        * 1. remove the element from HashMap

        *
        *
        */

       /*
        *  Important!!!!:
public V remove(Object key) {
       Entry<K,V> e = removeEntryForKey(key);
       return (e == null ? null : e.value);
   }

        *
        */
```

```
//      map.remove(rear);  // This is wrong.   should put key there.

        map.remove(rear.key);

//      System.out.println("remove:"+rear.key);
//      System.out.println(map.keySet());

        /*
         * 2. adjust the pointer
         *  (1) rear ==null
         *  (2) rear !=null
         */
        if(rear==null) {
            head=rear;
        }else{
            rear=rear.prev;

            /*
             * After you delete the last element:
             * (1) There are more than one elements are left.
             * (2) There is only one element left.  ---> no need to adjust the head pointer
             * (3) There is no element left ---> need to adjust the head pointer.
             *
             */

            // (1) There are more than one elements are left.
            if(rear!=null)  {
                rear.next=null;

            // (3) There is no element left ---> need to adjust the head pointer.
            }else{
                head=rear;
            }
        }


    }
```

Minimum Depth of Binary Tree

```
public static int minDepth2(TreeNode root) {

    boolean foundFlag = false;

    if (root == null)
        return 0;

    int result = 0;
    LinkedList<TreeNode> que = new LinkedList<TreeNode>();

    que.add(root);
    que.add(null);

    while (!que.isEmpty()) {

        TreeNode firstNode = que.pop();

        if (firstNode == null) {
            result++;
            if (foundFlag)
                return result;
            if (!que.isEmpty())
                que.addLast(null);

        } else {

            if (firstNode.left == null && firstNode.right == null) {
                foundFlag = true;
            }
            if (firstNode.left != null)
                que.addLast(firstNode.left);

            if (firstNode.right != null)
                que.addLast(firstNode.right);
        }

    }

    return result;
}
```

Maximum Depth of Binary Tree

BFS:

```
public static int maxDepthBFS(TreeNode root) {
    int len = 0;
    /*
     * Because we remove and add nodes so frequently, we choose LinkedList
     * as fundamental structure.
     */
    LinkedList<TreeNode> que = new LinkedList<TreeNode>();

    if (root != null) { // need to consider the special cases.
        que.add(root);
        que.add(null); // add a special mark to mark the first level
    }
    /*
     * 构造特征，并可以reuse，是必备的一种技能
     *
     * Creating new features based on the basic structure is a necessary
     * skill.
     *
     */

   while (!que.isEmpty()) {

        TreeNode cur = que.removeFirst(); // Thinking why put in? --->
                                          // should be in.

        if (cur == null) {
            len++; // if the que is empty, it means this level's nodes are
                   // visted.

            if (!que.isEmpty())
                que.addLast(null); // add the mark to mark this is the end
                                   // of this level.

        } else {
            if (cur.left != null)
                que.add(cur.left);
            if (cur.right != null)
                que.add(cur.right);
        }

    }

    return 0;
}
```

Another method:

```
public static int maxDepth(TreeNode node) {

    int left = 0, right = 0;

    if (node == null)
        return 0;
    if (node != null && node.left == null && node.right == null)
        return 1;

    if (node != null) {
        if (node.left != null) {
            left++;
            left += maxDepth(node.left);
        }

        if (node.right != null) {
            right++;
            right += maxDepth(node.right);

        }

        return Math.max(left, right);
    }

    return 0;

}
```

Another method 2:

```
public static int maxDeepbyDeepSearch(TreeNode root) {

    if (root == null)
        return 0;
    Stack<TreeNode> s = new Stack<TreeNode>();
    s.push(root);

    int maxDepth = 0;
    TreeNode prev = null;
    while (!s.empty()) {
        TreeNode curr = s.peek();

        if (prev == null || prev.left == curr || prev.right == curr) {
            if (curr.left != null)
                s.push(curr.left);
            else if (curr.right != null)
                s.push(curr.right);

        } else if (curr.left == prev) {
            if (curr.right != null)
                s.push(curr.right);
        } else {
            s.pop();
        }
        prev = curr;
        if (s.size() > maxDepth)
            maxDepth = s.size();
    }
    return maxDepth;
}
```

Binary Search

Implement Queue using stacks

```
public class ImplementQueueUsingStacks {

// Using two stack to implement the Queue
Stack<Integer> s1 = new Stack<Integer>();
Stack<Integer> s2 = new Stack<Integer>();

// Push element x to the back of queue.
public void push(int x) {
    s1.push(x);
}

// Removes the element from in front of queue.
public void pop() {
    if (s2.isEmpty()) {
        while (!s1.isEmpty()) {
            s2.push(s1.pop());
        }
    }
    // Because in the expression of the question: You may assume that all
    // operations are valid (for example, no pop or peek operations will be
    // called on an empty queue).
    // So in this place, we will not do any checking for empty.
    s2.pop();
}

// Get the front element.
public int peek() {
    if (s2.isEmpty()) {
        while (!s1.isEmpty()) {
            s2.push(s1.pop());
        }
    }
    // Because in the expression of the question: You may assume that all
    // operations are valid (for example, no pop or peek operations will be
    // called on an empty queue).
    // So in this place, we will not do any checking for empty.
    return s2.peek();
}

// Return whether the queue is empty.
public boolean empty() {

    return (s1.isEmpty() && s2.isEmpty()) ? true : false;
}

}
```

Implement Stack using Queues

```
    LinkedList<Integer> l1=new LinkedList<Integer>();
    LinkedList<Integer> l2=new LinkedList<Integer>();

   public void push(int x) {

       //always from l1, add elements
       l1.addLast(x);
   }

   // Removes the element on top of the stack.
   public void pop() {



       //handle only contain one element's case.
//         if(l1.size()==1) l1.removeFirst();

       int length=l1.size();

       for(int i=0;i<length-1;i++){
       int removeValue=l1.removeFirst();
       l2.addLast(removeValue);
//         System.out.println("remove:"+removeValue);
       }

//         l1.removeFirst();

       LinkedList<Integer> tmp=l1;
       l1=l2;
       l2=tmp;

       //always from l2 remove elements
       l2.removeFirst();
   }

   // Get the top element.
   public int top() {
//         System.out.println("top:"+l1.getLast());
       return l1.getLast();
   }

   // Return whether the stack is empty.
   public boolean empty() {
//         System.out.println(l1.isEmpty()&&l2.isEmpty());
       return (l1.isEmpty()&&l2.isEmpty());
   }
```

invert binary tree

```
/*
 * Accepted:
 *
 *
 */

public static TreeNode invertTree(TreeNode root){

    if(root==null||(root.left==null&&root.right==null)) return root;

    if(root!=null){

        if(root.left!=null||root.right!=null){
            TreeNode tmp=root.right;
            root.right=root.left;
            root.left=tmp;
        }

        if(root.left!=null)
            invertTree(root.left);

        if(root.right!=null)
            invertTree(root.right);
    }
    return root;
}
```

Swap two variables without using extra space

```
public static void main(String args[]){
    int a = 3;
    int b = 2;
    a = a ^ b;
    b = a ^ b;  //----> b= a^b^b ===> b=a
    a = a ^ b;  //----> a= a^a^b ===> a=b
    System.out.println(a + " " + b);
}
```

Merge two sorted arrays ===> array operation

Merge two sorted lists

O(1) time complexity to get minimum value of stack

```
Stack<Integer> s1 = new Stack<Integer>();
Stack<Integer> s2 = new Stack<Integer>();
int min = (int) (Math.pow(2, 32) - 1);

public void push(int x) {
    s1.push(x);

    if(!s2.isEmpty()) min=getMin();

    if (min > x) {
        min = x;
    }

    s2.push(min);

}

public void pop() {
    s1.pop();
    s2.pop();
    //if the statck is empty, should reset the min value to maximum.
    if(s2.isEmpty()){
     min = (int) (Math.pow(2, 32) - 1);
    }
}

public int top() {
    return s1.peek();
}

public int getMin() {
    return s2.peek();
}
```

Move-zeroes ===> array operation

```
public static void moveZeroes4(int[] nums) {
        makeFirstBeRight4(0,nums);
        for (int tmp : nums)
            System.out.print(tmp + " ");
}


public static void makeFirstBeRight4(int begin, int[] nums) {

    /*
     * Added this mark to terminate the useless all zero loop in advance
     *
     */


    //Do not use recursion method
//  // if the begin value is not zero
//  if (nums[begin] != 0 && begin + 1 < nums.length) {
//      makeFirstBeRight(begin + 1, nums);



    /*
     * (1) Have one pointer to indicate the first zero index in the array
     * (2) From the head, keep searching the zero until to the end of the array
```

```
     * (3) Once find zero, then, move this zero to after its position's
     * first find's non-zero number's position
     *
     */



    int p=begin;
    boolean stopflag=false;

    while(p!=nums.length-1&&!stopflag){
        for(int j=p;j<nums.length;j++){
            if(nums[j]==0) {
                p=j;
            break;
            }
        }

        //add one terminated condition
            //If cannot be able to find any 0 in the array, it is end.


            /*
    Input:
    [0,1,0,3,12]
    Output:
    [0,1,0,3,12]
    Expected:
    [1,3,12,0,0]

    Reason: use below sentence, actually it has defect.
             *
             *      if(p==begin) return;
             */


            //if the nums[p]!=0 and p==begin, that means it did not find any zero in array. it should return.
            if(nums[p]!=0&&p==begin) return;

            //add one terminated condition
            //If the p is reach the end already, it should return.
            if(p==nums.length-1) return;

            stopflag=true;



            // p is the current found 0's position

            for (int i = p+1; i < nums.length; i++) {
                if (nums[i] != 0) {
                    stopflag=false;
                    nums[p] = nums[i];
                    nums[i] = 0;
                    break;
                }
            }
        }

            //Do not use recursion method
    //      if (begin + 1 < nums.length&&!mark)
    //          makeFirstBeRight(begin + 1, nums);

    }
```

Palindrome Number (回文)

```
// reverse the number to see whether it is equal to previous number;
public boolean isPalindrome3(int x) {
    int tmp = x;
    int reverse = 0;

    while (tmp != 0) {
        reverse = reverse * 10 + (tmp % 10);
        tmp /= 10;
    }

    return reverse == x;
}

public boolean isPalindrome2(int x) {
    if (x < 0)
        return false;

    int div = 1;
    while (x / div >= 10) {
        div *= 10;
    }

    while (x != 0) {
        int l = x / div;
        int r = x % 10;
        if (l != r)
            return false;
        x = (x % div) / 10;
        div /= 100;
    }

    return true;
}
```

Permutations [排列 (important)]

method 1:

```
private static void permutation(String prefix, String str) {
    int n = str.length();
    if (n == 0) System.out.println(prefix);
    else {
        for (int i = 0; i < n; i++){
            System.out.println(prefix + str.charAt(i)+" || "+ str.substring(0, i) " || "+ str.substring(i+1, n));

            permutation(prefix + str.charAt(i), str.substring(0, i) + str.substring(i+1, n));
        }
    }
}
```

method 2: ----> int[] nums should contain non-repeated numbers

```
public void permutations(int[] nums,int i,int n){
    if(i==n){
        for(int tmp:nums)
        System.out.print(tmp+" ");

        System.out.println();
    }else{
        for(int j=i;j<nums.length;j++){
            swap(nums,i,j);
            permutations(nums,i+1,n);
            swap(nums,i,j);
        }
    }
}
```

Find path of Binary Tree

```
class TreeNode{
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){val=x;}
}

List<String> result=new ArrayList<String>();
List<TreeNode> list=new ArrayList<TreeNode>();

public void findPath(TreeNode root){

    if(root!=null){
        list.add(root);

        if(root.left==null&&root.right==null){
            String str="";
            for(TreeNode tmp:list)
            str+=tmp.val;
            result.add(str);
            list=new ArrayList<TreeNode>();
        }

        if(root.left!=null){
            findPath(root.left);
            list.remove(list.size()-1);
        }

        if(root.right!=null){
            findPath(root.right);
            list.remove(list.size()-1);
        }

    }

}
```

The sum of path:

```
boolean flag=false;
int sum=0;

public boolean hasPathSum(TreeNode root,int sum){

    if(root==null&&sum!=0) return false;
    if(root==null&&sum==0) return false;

    this.sum=sum;


    return loopTree(root,0);
}


public boolean loopTree(TreeNode root,int prevSum){


    //stop faster.
    if(flag) return true;

    if(root!=null){

        prevSum+=root.val;
//          System.out.println(root.val);


        if(root.left==null&&root.right==null) {
//              System.out.println(prevSum+"<-"+(prevSum==sum));
            if(prevSum==sum) flag=true;
        }

        if(root.left!=null) loopTree(root.left,prevSum);

//           result-=root.val;

         if(root.right!=null) loopTree(root.right,prevSum);
    }

    if(flag) return true;
 return false;

}
```

ReverseBits

```
public int reverse3(int n){

    // Convert the n to be 32 bits Binary number.
    StringBuilder nb = new StringBuilder(Integer.toBinaryString(n));
    nb.reverse();
    // for(int i=0;i<32-nb.length();i++){ ===> this is wrong. as nb.length
    // is changing. Be careful!!!!!!!!
    int tmpN = 32 - nb.length();
    for (int i = 0; i < tmpN; i++) {
        nb.append("0");
    }
    nb = nb.reverse();
    System.out.println(nb.toString());

    char[] nbchar=nb.toString().toCharArray();
    int head=0;
    int end=nb.length()-1;

    /*
     *   replace(old char,new char) this one will
     *   replace all the old char to new char.
    nbstr=nbstr.replace(nbstr.charAt(end), nbstr.charAt(head));
    nbstr=nbstr.replace(nbstr.charAt(head), tmp);
     */

while(head<end){
    char tmp=nbchar[end];
    nbchar[end]=nbchar[head];
    nbchar[head]=tmp;

    head++;
    end--;
}

System.out.println("final:"+String.valueOf(nbchar));

return (int)Long.parseLong(String.valueOf(nbchar),2);  //Used the long first and used (int) to cast the value. So it will not cause excep
tion.

}
```

Reverse Integer

```
public static int reverse2(int x) {
    if((Math.abs(x)+"").length()==1) return x;
    List<Integer> list=new ArrayList<Integer>();
    long result=0;

//      boolean flag=x>0?true:false;

    while(x!=0){
     list.add(x%10);
     x/=10;
    }

    for(int tmp:list){
        System.out.print(tmp+" ");
    }
    System.out.println();

    for(int i=0;i<list.size();i++){
        result+=list.get(i)*Math.pow(10,list.size()-i-1);

        System.out.print(list.size()-i-1+" ");
    }

    /*
     * Consider one issue: if the number is 100, then reverse it. The result should be 001?
     *
     */


    // Need to consider overflow(上溢) and underflow(下溢) at the same time.
//      if(result>Math.pow(2, 32)-1) return 0;

    /*
     * Java int:
     *  Minimum value is - 2,147,483,648.(-2^31) Maximum value is 2,147,483,647(inclusive).(2^31 -1)
     *
     */
    if(result>Math.pow(2, 31)-1||result<-Math.pow(2, 31)) return 0;

    System.out.println(result);

    return (int)result;
}
```

Reverse Linked List

```
public ListNode reverseList3(ListNode head) {
    if (head == null)
        return head;
    if (head.next == null)
        return head;
    if (head.next.next == null) {

        ListNode next = head.next;
        next.next = head;
        head.next = null;
        return next;
    }
    ListNode prev = head;

    head = head.next;
    ListNode next = head.next;

    prev.next = null;

    while (next.next != null) {
        head.next = prev;
        prev = head;
        head = next;
        next = next.next;
    }

    if (next.next == null) {
        head.next = prev;
    }

    next.next = head;

    return next;
}
```

Rotate Array

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

```
public static void rotate(int[] nums, int k) {

    int[] n=nums.clone();


    if(nums==null) return;

    for(int i=0;i<nums.length;i++){
        int tmp=n[i];

        nums[(i+k)%nums.length]=tmp;


    }
```

Reverse Array

Is Same Tree (/same-tree/)

```
public static boolean isSameTree(TreeNode p, TreeNode q) {

    ArrayList<TreeNode> list1 = new ArrayList<TreeNode>();
    ArrayList<TreeNode> list2 = new ArrayList<TreeNode>();
    getListFromTree(p, list1);
    getListFromTree(q, list2);

    if (list1.size() != list2.size())
        return false; // need to consider the length is different case.

    for (int i = 0; i < list1.size(); i++) {
        // Compare the value
        if (list1.get(i) != null && (list2.get(i) != null) && list1.get(i).val != list2.get(i).val)
            return false;

        // Compare the structure ---> use the null to distinguish differences.
        if ((list1.get(i) == null && list2.get(i) != null) || (list1.get(i) != null && list2.get(i) == null))
            return false;
    }

    return true;
}

public static void getListFromTree(TreeNode tree, ArrayList list) {

    if (tree == null)
        return;

    list.add(tree); // Do not forget the root element

    if (tree.left != null) {
        getListFromTree(tree.left, list);
        list.add(tree.left);
    } else {
        list.add(tree.left); // Do not forget null---> distinguish the
                             // structure
    }

    /*
     * In order to make the logic clear to read, keep the above code
     *
     * Actually, it can be replaced below:
     *
     * if(tree.left!=null) getListFromTree(tree.left,list);
     * list.add(tree.left);
     *
     *
     */

    if (tree.right != null) {
        getListFromTree(tree.right, list);
        list.add(tree.right);
    } else {
        list.add(tree.right); // Do not forget null---> distinguish the
                              // structure
    }

    /*
     * In order to make the logic clear to read, keep the above code
     *
     * Actually, it can be replaced below:
     *
     * if(tree.right!=null) getListFromTree(tree.right,list);
     * list.add(tree.right);
     *
     *
     */

}
```

Symmetric Tree

```
public static boolean isSymmetric2(TreeNode root) {

    if(root==null) return true;

    return symmSubTree(root.left,root.right);
}


public static boolean symmSubTree(TreeNode left, TreeNode right){

    if(left==null&&right==null) return true;
    if(left!=null&&right==null) return false;
    if(left==null&&right!=null) return false;

    if(left.val!=right.val) return false;

    if(!symmSubTree(left.left,left.right)) return false;
    if(!symmSubTree(right.right,right.left)) return false;

    return true;

}
```

valid-anagram (/valid-anagram/)

Given two strings s and t, write a function to determine if t is an anagram of s.

For example, s = "anagram", t = "nagaram", return true. s = "rat", t = "car", return false.

Note: You may assume the string contains only lowercase alphabets.

Follow up: What if the inputs contain unicode characters? How would you adapt your solution to such case?

```
public static boolean isAnagram(String s, String t) {

    if (s.length() != t.length())
        return false;

    HashMap<Character, Integer> sMap = new HashMap<Character, Integer>();
    HashMap<Character, Integer> tMap = new HashMap<Character, Integer>();

    char[] sCharArr = s.toCharArray();
    char[] tCharArr = t.toCharArray();

    for (Character tmp : sCharArr) {
        if (sMap.get(tmp) == null)
            sMap.put(tmp, 1);
        else {
            sMap.put(tmp, sMap.get(tmp) + 1);
        }

    }

    for (Character tmp : tCharArr) {
        if (tMap.get(tmp) == null)
            tMap.put(tmp, 1);
        else {
            tMap.put(tmp, tMap.get(tmp) + 1);
        }
    }

    for (Character tmp : tCharArr) {
        if (sMap.get(tmp) == null || tMap.get(tmp) == null){
            System.out.println(1);
            return false;
        }
        else {
            if (sMap.get(tmp).intValue()!= tMap.get(tmp).intValue()){
                return false;
            }
        }

    }

    for (Character tmp : sCharArr) {
        if (sMap.get(tmp) == null || tMap.get(tmp) == null){
            return false;
        }
        else {
            if (sMap.get(tmp).intValue()!= tMap.get(tmp).intValue()){

                return false;
            }
        }
    }

    return true;

}
```

HashSet iteration

```
    HashSet set=new HashSet();

    set.add("123");

    for(Iterator it=set.iterator();it.hasNext();){
        System.out.println(it.next());
    }
```

HashMap iteration

Method 1: (method 1 is better than method 2----> entry is the basic type in HashMap's implementation.)

```
for(Entry<Integer, String> entry:map.entrySet())
  {
   System.out.println(entry.getKey()+"="+entry.getValue());
  }
```

Method 2:

```
for(Object obj : map.keySet()) {
      Object key = obj;
      Object value = map.get(obj);
      System.out.println(value);
  }
```

HashTable iteration

```
Hashtable table = new Hashtable();
  table.put(1, "1");
  table.put(2, "1");
  table.put(3, "1");
  //遍历key
  Enumeration e = table.keys();

      while( e. hasMoreElements() ){

      System.out.println( e.nextElement() );

      }
  //遍历value
  e = table.elements();

  while( e. hasMoreElements() ){

  System.out.println( e.nextElement() );

  }
```

```
Hashtable table=new Hashtable()

table.put("dfs","fds");

Enumeration keys=table.keys();

while(keys.hasMoreElements()){
    str=(String) keys.nextElement();
    System.out.println(str+":"+table.get(str));
}
```

MaximumSubarray

```
public int maxSubArray3(int[] nums) {
    int max_ending_here=nums[0];
    int max_so_far=nums[0];

    for(int i=1;i<nums.length;i++){
        max_ending_here=Math.max(nums[i], max_ending_here+nums[i]);
        max_so_far=Math.max(max_so_far,max_ending_here);
    }

    return max_so_far;
}
```

LinkRightNode

Link all the same level node from left to right by using "Right" field.

```
class Node
{
    public Node[] Children;
    public Node Right;
    public int val;
    Node(int x){val=x;}
}


public Node linkRightNode(Node rootNode){

    if(rootNode==null) return null;  //consider the null case.

    LinkedList<Node> que=new LinkedList<Node>();

    que.addLast(rootNode);
    que.addLast(null);

    Node prev=null;

    while(!que.isEmpty()){
        Node firstNode=que.removeFirst();

        if(firstNode==null){
            prev=null;
            if(!que.isEmpty()){
                que.addLast(null);
            }
        }else{

            if(prev!=null){
                prev.Right=firstNode;
            }

            if(firstNode.Children!=null)
            for(Node tmp:firstNode.Children){
                que.addLast(tmp);
            }
            prev=firstNode;
        }
    }
    return rootNode;
}
```

反转输出：

I love computer to i evol retupmoc.

```
public static void main(String args[]) {
    String str = "I love shopee !";
    StringBuilder sb = new StringBuilder();
    String[] str_arr = str.split(" ");
    for (String tmp : str_arr) {
        sb.append(reverseStr(tmp) + " ");
    }
    System.out.println(sb.toString());
}
public static String reverseStr(String input) {
    String str = input;
    if (input == null) {
        return null;
    }
    if (input != null && input.length() <= 1) {
        return input;
    }
    char[] char_arr = str.toCharArray();
    for (int i = 0; i <= (input.length() - 1) / 2; i++) {
        // swap the value
        char tmp = char_arr[i];
        char_arr[i] = char_arr[input.length() - 1 - i];
        char_arr[input.length() - 1 - i] = tmp;
    }
    return String.valueOf(char_arr);
}
```