简单：

1、HashMap总结

* java中hashmap常用函数总结：

getOrDefault（Object key, V defaultValue）：如果哈希表中有key的时候，就使用这个key值，如果没有的话，就是用默认的defaultValue;

put(Object key,Object value):  
get():通过键的方式进行值的读取  
put（K key，V value）：将键和对应的哈希值插入到hashmap中

2、有效的字母异位词  
解法一：//通过调用排序函数

class Solution {

public boolean isAnagram(String s, String t) {

char[] sstr=s.toCharArray();

char[] tstr=t.toCharArray();

Arrays.sort(sstr);

Arrays.sort(tstr);

return String.valueOf(sstr).equals(String.valueOf(tstr));

}

}

解法二：//构建一个长度为26的数组，用于存储字符串中字母出现的次数，使用数组下标//表示是哪一个字母。

class Solution {

public boolean isAnagram(String s, String t) {

if (s.length() != t.length()) {

return false;

}

int[] table = new int[26];

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

table[s.charAt(i) - 'a']++;

}

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

table[t.charAt(i) - 'a']--;

if (table[t.charAt(i) - 'a'] < 0) {

return false;

}

}

return true;

}

}

//进阶：如果存在unicode字符的话，直接使用一个HashMap维护字符串中字符出现的频次，//然后对另一个字符串进行遍历，对该字符串中的字符次数如果存在的话，就进行减少。当//减少之后哈希表中的次数小于0的时候，则返回false.

1. 两数之和

解法一：暴力

class Solution {

public int[] twoSum(int[] nums, int target) {

int[] res=new int[2];

for(int i=0;i<nums.length-1;i++){

for(int j=i+1;j<nums.length;j++){

if((nums[i]+nums[j])==target){

res[0]=i;

res[1]=j;

return res;

}

}

}

return res;

}

}

解法二：哈希表

class Solution {

public int[] twoSum(int[] nums, int target) {

int[] res=new int[2];

//使用哈希表的方法实现

Map<Integer,Integer> hashtable=new HashMap<Integer,Integer>();

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

if(hashtable.containsKey(target-nums[i])){

return new int[]{hashtable.get(target-nums[i]),i};

}

hashtable.put(nums[i],i);

}

return new int[0];

}

}

1. N叉树的前序遍历  
   class Solution {

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

public List<Integer> preorder(Node root) {

order(root);

return res;

}

public void order(Node root){

if(root==null){

return;

}

res.add(root.val);

for(int i=0;i<root.children.size();i++){

order(root.children.get(i));

}

}

}

1. 自学HeapSort:

中等：  
1、字母异位词分组  
解法：这个题目的难点在于找到key是什么，value是什么。  
class Solution{

public List<List<String>> groupAnagrams(String[] strs){

If(strs.length==0) return new ArrayList();

Map<String,List> ans=new HashMap<String,List>();

for(String s:strs){

char[] ca=s.toCharArray();

Arrays.sort(ca);

String key=String.valueOf(ca);

If(!ans.containsKey(key)) ans.put(key,new ArrayList());

ans.get(key).ans(s);

}

Return new ArrayList(ans.values());

}

}

1. 二叉树的中序遍历  
   递归实现：

class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

inorder(root,res);

return res;

}

public void inorder(TreeNode root,List<Integer> res){

if(root==null){

return;

}

inorder(root.left,res);

res.add(root.val);

inorder(root.right,res);

}

}

1. 二叉树的前序遍历

class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

inorder(root,res);

return res;

}

public void preorder(TreeNode root,List<Integer> res){

if(root==null){

return;

}  
res.add(root.val);

preorder(root.left,res);

preorder(root.right,res);

}

}

1. N叉树的层序遍历  
   class Solution{  
    private List<List<Integer>> result=new ArrayList<>();

public List<List<Integer>> levelOrder(Node root){

If(root!=null) traverseNode(root,0);

return result;

}

private void traverseNode(Node node, int level){

If(result.size()<=level){

result.add(new ArrayList<>());

}

result.get(level).add(node.val);

for(Node child:node.children){

traverseNode(child,level+1);

}

}

}

5、丑数  
class Ugly {

public int[] nums = new int[1690];

Ugly() {

HashSet<Long> seen = new HashSet();

PriorityQueue<Long> heap = new PriorityQueue<Long>();

seen.add(1L);

heap.add(1L);

long currUgly, newUgly;

int[] primes = new int[]{2, 3, 5};

for(int i = 0; i < 1690; ++i) {

currUgly = heap.poll();

nums[i] = (int)currUgly;

for(int j : primes) {

newUgly = currUgly \* j;

if (!seen.contains(newUgly)) {

seen.add(newUgly);

heap.add(newUgly);

}

}

}

}

}

class Solution {

public static Ugly u = new Ugly();

public int nthUglyNumber(int n) {

return u.nums[n - 1];

}

}

//没太理解

6、前k个高频元素  
class Solution {

public int[] topKFrequent(int[] nums, int k) {

Map<Integer, Integer> occurrences = new HashMap<Integer, Integer>();

for (int num : nums) {

occurrences.put(num, occurrences.getOrDefault(num, 0) + 1);

}

// int[] 的第一个元素代表数组的值，第二个元素代表了该值出现的次数

PriorityQueue<int[]> queue = new PriorityQueue<int[]>(new Comparator<int[]>() {

public int compare(int[] m, int[] n) {

return m[1] - n[1];

}

});

for (Map.Entry<Integer, Integer> entry : occurrences.entrySet()) {

int num = entry.getKey(), count = entry.getValue();

if (queue.size() == k) {

if (queue.peek()[1] < count) {

queue.poll();

queue.offer(new int[]{num, count});

}

} else {

queue.offer(new int[]{num, count});

}

}

int[] ret = new int[k];

for (int i = 0; i < k; ++i) {

ret[i] = queue.poll()[0];

}

return ret;

}

}

//利用哈希表记录每个数字出现的次数，形成一个出现次数的数组，再利用堆的思想，构建//一个小顶堆，然后遍历出现次数数组，如果堆的元素个数小于k，就可以直接插入堆中，//如果堆的元素个数小于k，则检查堆顶与当前出现次数的大小，如果堆顶更大的话，说明//至少k个数字的出现比当前值更大，所以应当舍弃当前值，否则就弹出堆顶，并将当前值//插入堆中。