第1题：1000000转换格式为1000.000

**package** test;

**import** java.util.Scanner;

**public** **class** format {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

//String str="1000";

Scanner n = **new** Scanner(System.***in***);

String str=n.next();

StringBuffer sb=**new** StringBuffer();

**char**[] charstr=str.toCharArray();

**for**(**int** i=charstr.length-1;i>=0;)

{

**if**(i>=3)

{

sb.append(charstr[i]);

sb.append(charstr[i-1]);

sb.append(charstr[i-2]);

sb.append(",");

i=i-3;

}

**else**

{

sb.append(charstr[i]);

i--;

}

}

System.***out***.println(sb.reverse());

}

}

第二题：

整形数组***升序排列***，去重。空间复杂度O(1)，时间复杂度：O(N)

Set去重：

1. String[] array = {"a","b","c","c","d","e","e","e","a"};
2. Set<String> set = **new** HashSet<>();
3. **for**(**int** i=0;i<array.length;i++){
4. set.add(array[i]);
5. }
6. String[] arrayResult = (String[]) set.toArray(**new** String[set.size()]);
7. System.out.println(Arrays.toString(arrayResult));

int f(int arr[], int num)  
{  
  int i = 0;  
  for(int j = 1; j < num; ++j)  
    if(arr[i] != arr[j])  
      arr[++i] = arr[j];  
  return i+1;  
}  
  
已测试通过。

二面：平衡二叉树的特点，二叉搜索树的特点。

原题 　　Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

题目大意

　给定一个升序排列的二叉树，将其转换为一棵高度平衡的二叉树。 解题思路 　　采用递归分治法。

代码实现

树结点类

public class TreeNode

{

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) { val = x; }

}

算法实现类

public class Solution

{

public TreeNode sortedArrayToBST(int[] nums)

{

// 参数检验

if (nums == null || nums.length < 1)

{

return null;

}

// 递归分治法求解

return solve(nums, 0, nums.length - 1);

}

/\*\*

\* 递归分治求解方法

\*

\* @param nums 升序排序数组

\* @param start 开始位置

\* @param end 结束位置

\* @return 根结点

\*/

public TreeNode solve(int[] nums, int start, int end)

{

// 还有未处理的数据

if (start <= end)

{

// 找蹭位置

int mid = start + ((end - start) >> 1);

// 构造根结点

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

// 求左子树

root.left = solve(nums, start, mid - 1);

// 求右子树

root.right = solve(nums, mid + 1, end);

// 返回结果

return root;

}

return null; }

}

第二题：map.values()用法

原题：

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],   
Return:

[

["ate", "eat","tea"],

["nat","tan"],

["bat"]

]

题目大意：给出一个字符串数组，将其按照组成字母相同这一条件来分组，即组成字符串的字符完全相同的字符串分成一组。

解题思路：

         采用HashMap<String,List<String>>，每次讲读入的字符串在map中查找（这里需将读入的字符串转化成数组后用sort（）来排列好）。

1. **public** **class** Solution {
2. **public** List<List<String>> groupAnagrams(String[] strs) {
3. List<List<String>> list = **new** ArrayList<List<String>>();
4. **int** len = strs.length;
5. **if**(len<1) **return** list;
6. Map<String,List<String>> map = **new** HashMap<String,List<String>>();
7. String tmp = "";
8. **for**(**int** i=0;i<len;i++){
9. tmp = strs[i];
10. **char**[] arrayOfString = tmp.toCharArray();
11. Arrays.sort(arrayOfString);
12. tmp = **new** String(arrayOfString);
13. **if**(map.containsKey(tmp)){
14. map.get(tmp).add(strs[i]);
15. }**else**{
16. List<String> item = **new** ArrayList<String>();
17. item.add(strs[i]);
18. map.put(tmp, item);
19. }
20. }
21. **for** (List<String> value : map.values()) {
22. list.add(value);
23. }
24. **return** list;
25. }
26. }