<http://www.shucunwang.com/RunCode/java/>

算法：

**1、Reverse a singly lined list**

1）Collections.reverse()

public static void main(Strings[] args)

{

String[] a = {"q","w","e","r","t"};

System.out.println("original:"+Arrays.asList(arrays));

List<String> tmp = Arrays.asList(arrays);

Collections.reverse(tmp);

Sysout(tmp);

}

2）traverse from tail to head: O(n)

public static void main(Strings[] args)

{

String[] a = {"q","w","e","r","t"};

System.out.println("original:"+Arrays.asList(arrays));

reverse(a);

}

public static String[] reverse(String[] a)

{

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

{

return a;

}

String[] reversedArray = new String[a.length];

for(int i=a.length-1,j=0; i>=0;i--,j++)

{

reversedArray[j] = a[i];

}

return reversedArray;

}

3) change the head and tail, change the next of head and the previous of tailO(n/2)

public static String[] reverse(String[] a)

{

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

{

return a;

}

int i=0;

int j=a.length-1;

for(int i=0,j=a.length-1;i<=a.length/2&&j>=a.length;i++,j--)

{

swap(a, i, j);

}

return reversedArray;

}

public static void swap(String[] a, int i, int j)

{

String tmp = a[i];

a[i] = a[j];

a[j] = tmp;

}

**2、Print a binary search tree level by level**

public class BsTreeNode

{

BsTreeNode left;

BsTreeNode right;

BsTreeNode parent;

int value;

//pre\_order:

public void pre\_order(BsTreeNode node)

{

if(node==null)

{

return node;

}else{

sysout(node.value)

pre\_order(node.left);

pre\_order(node.right);

}

}

}

**3、Max sub sequence problem**

只求最大和：

public static void main(Strings[] args)

{

int[] n;

int maxSum=0;

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

{

int currentSum = 0;

for(int j=i;j<n.length;j++)

{

currentSum+=n[j];

if(currentSum>maxSum)

{

maxSum=currentSum;

}

}

}

sysout(maxSum);

}

//线性的算法O(N)

long maxSubSum4(const vector<int>& a)

{

       long maxSum = 0, thisSum = 0;

       for (int j = 0; j < a.size(); j++)

       {

              thisSum += a[j];

              if (thisSum > maxSum)

                     maxSum = thisSum;

              else if (thisSum < 0)

                     thisSum = 0;

       }

       return maxSum;

}

**3、Find the nth smallest/largest number in a unsorted array**

**4、Two sum, three sun problem**

**5、Java pow**

**double** pow(**int** a,**int** b){

**double** yourNum =1;

**for**(**int** i=0; i< b; i++)

 **if**(b>=0)

   yourNum\*=a;

  **else**

    yourNum/=a;

 **return** yourNum;

 }

**6、if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.**

1、冒泡

**public** **static** **void** bubbleRank2(**int** n[]){

               **for** (**int** i =0; i <n .length ;i ++)

                      **for**(**int** j =1;j <n .length -i;j ++)

                     {

                            **if**(n [j-1 ]>n [j ]){

swap(n[j-1], n[j]);

                           }

                     }

       }

**public** **static** **void** bubbleRank(**int** n[]){

               **for** (**int** i =0; i <n .length ;i ++)

                      **for**(**int** j =0;j <n .length -i -1;j ++)

                     {

                            **if**(n [j ]>n [j +1]){

                                   **int** tmp = n [j ];

                                   n[ j]= n[ j+1];

                                   n[ j+1]= tmp;

                           }

                     }

       }

2、快排

**public** **static** **void** quicksort(**int** n[], **int** left, **int** right) {

               **int** dp ;

               **if** (left < right ) {

                      dp = *partition*(n, left, right);

                      *quicksort*(n, left, dp - 1);

                      *quicksort*(n, dp + 1, right);

              }

       }

        **static** **int** partition (**int** n [], **int** left, **int** right) {

               **int** pivot = n [left ];

               **while** (left < right ) {

                      **while** (left < right && n[right] >= pivot)

                            right--;

                      **if** (left < right )

                            n[ left++] = n[ right];

                      **while** (left < right && n[left] <= pivot)

                            left++;

                      **if** (left < right )

                            n[ right--] = n[ left];

              }

               n[ left] = pivot;

               **return** left ;

       }

3、直接插入排序straight insertion sort

increase:

**public** **static** **void** main(**int** n[], **int** left, **int** right) {

int[] n=[1,5,2,6...];

straightSort(n);

}

public static straightSort(n)

{

for(int i=2;i<n.length;i++)

{

if(n[i]<n[i-1])

{

n[0]=n[i];

n[i]=n[i-1];

for(j=i-2;n[0]<n[j];j--)

{

n[j+1]=j[j];

n[j+1]=n[0];

}

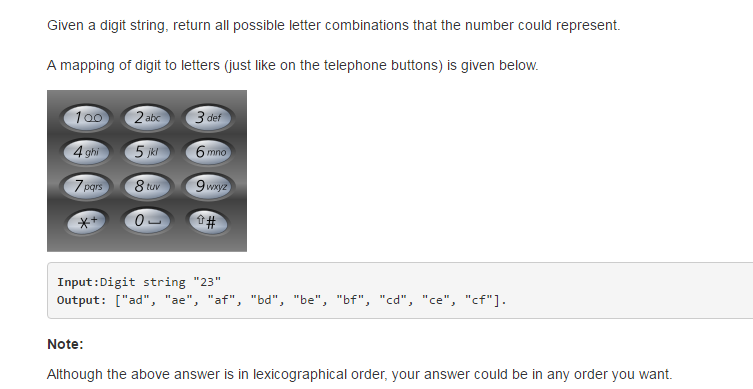
}

}

}

算法：

1、Letter Combinations of a Phone Number



解：

public static void main(String[] args)

{

Solution solution = new Solution();

List<String> result = null;

result = solution.letterCombinations("345");

System.out.println(result);

}

public List<String> letterCombinations(String digits) {

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

Map<Integer, String> telMap = new HashMap<Integer, String>();

telMap.put(2, "abc");

telMap.put(3, "def");

telMap.put(4, "ghi");

telMap.put(5, "jkl");

telMap.put(6, "mno");

telMap.put(7, "pqrs");

telMap.put(8, "tuv");

telMap.put(9, "wxyz");

//validate the input

if(null==digits||digits.length()==0)

{

return result;

}

result = getCombinations(result, digits, telMap);

return result;

}

private List<String> getCombinations(List<String> original, String subDigits, Map<Integer, String> telMap)

{

List<String> tmpRes = new ArrayList<String>();

if(null==subDigits||subDigits.length()==0)

{

return original;

}else

{

int currentNum = Integer.parseInt(subDigits.charAt(0)+"");

String currentStr = telMap.get(currentNum);

if(null!=currentStr)

{

if(original.isEmpty())

{

for(int j=0;j<currentStr.length();j++)

{

tmpRes.add(currentStr.charAt(j)+"");

}

}else{

for(int i=0;i<original.size();i++)

{

for(int j=0;j<currentStr.length();j++)

{

String oldStr = original.get(i);

String newStr = oldStr.concat(currentStr.charAt(j)+"");

tmpRes.add(newStr);

}

}

}

}

return getCombinations(tmpRes, subDigits.substring(1, subDigits.length()), telMap);

}

}

2、Generate Parentheses

Given *n* pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given *n* = 3, a solution set is:

[
"((()))",
"(()())",
"(())()",
"()(())",
"()()()"
]

解：

public static void main(String[] args)

{

Solution solution = new Solution();

List<String> result = null;

result = solution.generateParenthesis(3);

System.out.println(result);

}

public List<String> generateParenthesis(int n) {

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

//use x,y to record the numbers of parenthesis left

int left=0;

int right=0;

String res = "";

addParenthesis(result, res, left, right, n);

return result;

}

private void addParenthesis(List<String> result, String res, int left, int right, int n)

{

if(res.length()==n\*2)

{

result.add(res);

return;

}

if(left < n)

{

addParenthesis(result, res+"(", left+1, right, n);

}

if(right<left)

{

addParenthesis(result, res+")", left, right+1, n);

}

}

3、Remove Duplicate Letters（）

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

**Example:**

Given "bcabc"

Return "abc"

Given "cbacdcbc"

Return "acdb"

解：

**public** String **removeDuplicateLetters**(String s) {
**int**[] cnt = **new** **int**[26];
**int** pos = 0; // the position for the smallest s[i]
**for** (**int** i = 0; i < s.length(); i++) cnt[s.charAt(i) - 'a']++;
**for** (**int** i = 0; i < s.length(); i++) {
**if** (s.charAt(i) < s.charAt(pos)) pos = i;
**if** (--cnt[s.charAt(i) - 'a'] == 0) **break**;
}
**return** s.length() == 0 ? "" : s.charAt(pos) + removeDuplicateLetters(s.substring(pos + 1).replaceAll("" + s.charAt(pos), ""));
}

4、Minimum Genetic Mutation

A gene string can be represented by an 8-character long string, with choices from "A", "C", "G", "T".

Suppose we need to investigate about a mutation (mutation from "start" to "end"), where ONE mutation is defined as ONE single character changed in the gene string.

For example, "AACCGGTT" -> "AACCGGTA" is 1 mutation.

Also, there is a given gene "bank", which records all the valid gene mutations. A gene must be in the bank to make it a valid gene string.

Now, given 3 things - start, end, bank, your task is to determine what is the minimum number of mutations needed to mutate from "start" to "end". If there is no such a mutation, return -1.

**Note:**

Starting point is assumed to be valid, so it might not be included in the bank.

If multiple mutations are needed, all mutations during in the sequence must be valid.

You may assume start and end string is not the same.

**Example 1:**

start: "AACCGGTT"
end: "AACCGGTA"
bank: ["AACCGGTA"]
return: 1

**Example 2:**

start: "AACCGGTT"
end: "AAACGGTA"
bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]
return: 2

**Example 3:**

start: "AAAAACCC"
end: "AACCCCCC"
bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]
return: 3

解：