# 3年JAVA技能总结-算法篇

1. 斐波那契数列

递归

private static long fibonacci(int n) {

if (n == 1 || n == 2) {

return 1;

}

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

}

for循环

private static long fibonacci2(int n) {

if (n == 1 || n == 2) {

return 1;

}

long f1 = 1L, f2 = 1L;

long fn = 0L;

while (n > 2) {

fn = f1 + f2;

f1 = f2;

f2 = fn;

n--;

}

return fn;

}

尾递归

private static long Fib(int n) {

if (n < 2) return n;

return fibonacci3(n, 1, 1, 3);

}

1. 最大公约数和最小公倍数

质因数分解法：

短除法：

辗转相除法：

更相减损法：

**public class** 最大公约数和最小公倍数 {  
  
 */\*\*  
 \* 最大公约数和最小公倍数：两个自然数的最大公约数与它们的最小公倍数的乘积等于这两个数的乘积  
 \* 质因数分解法：  
 \* 把每个数分别分解质因数，再把各数中的全部公有质因数提取出来连乘，所得的积就是这几个数的最大公约数  
 \* 例如：求24和60的最大公约数，先分解质因数，得24=2×2×2×3，60=2×2×3×5，24与60的全部公有的质因数是2、2、3，它们的积是2×2×3=12，所以，（24，60）=12  
 \* 把几个数先分别分解质因数，再把各数中的全部公有的质因数和独有的质因数提取出来连乘，所得的积就是这几个数的最小公倍数  
 \* 例如：求6和15的最小公倍数。先分解质因数，得6=2×3，15=3×5，6和15的全部公有的质因数是3，6独有质因数是2，15独有的质因数是5，2×3×5=30，  
 \* 30里面包含6的全部质因数2和3，还包含了15的全部质因数3和5，且30是6和15的公倍数中最小的一个，所以[6，15]=30  
 \*  
 \* 短除法：  
 \* 短除法求最大公约数，先用这几个数的公约数连续去除，一直除到所有的商互质为止，然后把所有的除数连乘起来，所得的积就是这几个数的最大公约数  
 \* 求最大公因数便乘一边，求最小公倍数便乘一圈。  
 \* 短除法的本质就是质因数分解法，只是将质因数分解用短除符号来进行  
 \* 短除符号就是除号倒过来。短除就是在除法中写除数的地方写两个数共有的质因数，然后落下两个数被公有质因数整除的商，之后再除，以此类推，直到结果互质为止（两个数互质）。  
 \* 而在用短除计算多个数时，对其中任意两个数存在的因数都要算出，其它没有这个因数的数则原样落下。直到剩下每两个都是互质关系。  
 \* 无论是短除法，还是分解质因数法，在质因数较大时，都会觉得困难。这时就需要用新的方法  
 \*  
 \* 辗转相除法：求两个自然数的最大公约数的一种方法，也叫欧几里德算法  
 \* 377 ÷ 319 =1...58  
 \* 319 ÷ 58 =5...29  
 \* 58 ÷ 29 =2 ∴（319，377）=29  
 \* 用辗转相除法求几个数的最大公约数，可以先求出其中任意两个数的最大公约数，再求这个最大公约数与第三个数的最大公约数，依次求下去，直到最后一个数为止。  
 \* 最后所得的那个最大公约数，就是所有这些数的最大公约数  
 \*  
 \* 更相减损法：  
 \* 第一步：任意给定两个正整数；判断它们是否都是偶数。若是，则用2约简；若不是则执行第二步。  
 \* 第二步：以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。继续这个操作，直到所得的减数和差相等为止。  
 \* 则第一步中约掉的若干个2与第二步中等数的乘积就是所求的最大公约数  
 \* 其中所说的“等数”，就是最大公约数。求“等数”的办法是“更相减损”法。所以更相减损法也叫等值算法。  
 \*  
 \*/* @Test  
 **public void** method1() {  
 System.***out***.println(**"质因数分解法：\n"** +  
 **" 把每个数分别分解质因数，再把各数中的全部公有质因数提取出来连乘，所得的积就是这几个数的最大公约数\n"** +  
 **" 例如：求24和60的最大公约数，先分解质因数，得24=2×2×2×3，60=2×2×3×5，24与60的全部公有的质因数是2、2、3，它们的积是2×2×3=12，所以，（24，60）=12\n"** +  
 **" 把几个数先分别分解质因数，再把各数中的全部公有的质因数和独有的质因数提取出来连乘，所得的积就是这几个数的最小公倍数\n"** +  
 **" 例如：求6和15的最小公倍数。先分解质因数，得6=2×3，15=3×5，6和15的全部公有的质因数是3，6独有质因数是2，15独有的质因数是5，2×3×5=30，\n"** +  
 **" 30里面包含6的全部质因数2和3，还包含了15的全部质因数3和5，且30是6和15的公倍数中最小的一个，所以[6，15]=30"**);  
 **int** a = 60, b = 30;  
 List<Integer> a1 = getPrimeFactors(a);  
 List<Integer> b1 = getPrimeFactors(b);  
  
 *//得到他们的公有质因数：  
 //求积* a1.retainAll(b1);  
 **int** max = 1, min = 1;  
 **for** (Integer x : a1) {  
 **if** (b1.contains(x)) {  
 b1.remove(x);  
 max \*= x;  
 } **else** {  
 min \*= x;  
 }  
  
 }  
  
 System.***out***.println(**"最大公因数："** + max);  
  
*/\* int quadrature2 = gcd;  
 for (Integer x : a1) {  
 if (!commonFactorList.contains(x)) {  
 quadrature2 \*= x;  
 }  
 }  
 for (Integer x : b1) {  
 if (!commonFactorList.contains(x)) {  
 quadrature2 \*= x;  
 }  
 }\*/* System.***out***.println(**"最小公倍数："** + min);  
 }  
  
 */\*\*  
 \** ***@param num*** *\** ***@return*** *存储质因子序列的数组  
 \** ***@描述*** *返回一个数的质因子序列, 这里不考虑 1  
 \*/* **private** List<Integer> getPrimeFactors(**int** num) {  
 List<Integer> factorList = **new** ArrayList<>();  
  
 num = Math.*abs*(num);  
 **int** i = 2;  
 **int** k = 2;  
 **while** (num >= k) {  
 **if** (num == k) {  
 factorList.add(k);  
 **break**;  
 } **else if** (num % k == 0) {  
 factorList.add(k);  
 num = num / k;  
 } **else** {  
 k++;  
 }  
 }  
  
 */\*int start = 2;  
 while (start < num) {  
 if (num % start == 0) {  
 factorList.add(start);  
 num /= start;  
 // start = 2;  
 } else {  
 start++;  
 }  
 }  
 factorList.add(num); //////////////\*/* **return** factorList;  
 }  
  
  
 @Test  
 **public void** method2() {  
 System.***out***.println(**"短除法：\n"** +  
 **" 短除法求最大公约数，先用这几个数的公约数连续去除，一直除到所有的商互质为止，然后把所有的除数连乘起来，所得的积就是这几个数的最大公约数\n"** +  
 **" 求最大公因数便乘一边，求最小公倍数便乘一圈。\n"** +  
 **" 短除法的本质就是质因数分解法，只是将质因数分解用短除符号来进行\n"** +  
 **" 短除符号就是除号倒过来。短除就是在除法中写除数的地方写两个数共有的质因数，然后落下两个数被公有质因数整除的商，之后再除，以此类推，直到结果互质为止（两个数互质）。\n"** +  
 **" 而在用短除计算多个数时，对其中任意两个数存在的因数都要算出，其它没有这个因数的数则原样落下。直到剩下每两个都是互质关系。\n"** +  
 **" 无论是短除法，还是分解质因数法，在质因数较大时，都会觉得困难。这时就需要用新的方法"**);  
 **int** a = 24, b = 240;  
 **int** gcd = 1;  
 **for** (**int** i = 2; i <= Math.*min*(a, b); i++) {  
 **if** (a % i == 0 && b % i == 0) {  
 gcd \*= i;  
 a = a / i;  
 b = b / i;  
 i--;  
 }  
 }  
 System.***out***.println(**"最大公因数："** + gcd);  
 System.***out***.println(**"最小公倍数："** + (gcd \* a \* b));  
 }  
  
 @Test  
 **public void** method3() {  
 System.***out***.println(**"辗转相除法：求两个自然数的最大公约数的一种方法，也叫欧几里德算法\n"** +  
 **" 377 ÷ 319 =1...58\n"** +  
 **" 319 ÷ 58 =5...29\n"** +  
 **" 58 ÷ 29 =2 ∴（319，377）=29\n"** +  
 **" 用辗转相除法求几个数的最大公约数，可以先求出其中任意两个数的最大公约数，再求这个最大公约数与第三个数的最大公约数，依次求下去，直到最后一个数为止。\n"** +  
 **" 最后所得的那个最大公约数，就是所有这些数的最大公约数"**);  
 **int** a = 60, b = 24;  
 **int** quadrature = a \* b;  
 **int** gcd = 1, remainder;  
 **for** (; ; ) {  
 **int** max = Math.*max*(a, b);  
 **int** min = Math.*min*(a, b);  
 remainder = max % min;  
 **if** (remainder == 0) {  
 gcd = min;  
 **break**;  
 }  
 a = min;  
 b = remainder;  
 }  
 System.***out***.println(**"最大公因数："** + gcd);  
 System.***out***.println(**"最小公倍数："** + (quadrature / gcd));  
 }  
  
 */\*\*  
 \* 辗转相除法（欧几里得算法）  
 \* 思路：取两个数中最大的数做除数，较小的数做被除数，用最大的数除较小数，如果余数为0，则较小数为这两个数的最大公约数，如果余数不为0，用较小数除上一步计算出的余数，直到余数为0，则这两个数的最大公约数为上一步的余数。  
 \* 1、递归  
 \*  
 \** ***@param a*** *\** ***@param b*** *\** ***@return*** *\*/* **private static int** gcd(**int** a, **int** b) {  
 **return** (b == 0) ? a : *gcd*(b, a % b);  
 }  
  
 */\*\*  
 \* 辗转相除法（欧几里得算法）  
 \* 2、非递归形式  
 \*  
 \** ***@param a*** *\** ***@param b*** *\** ***@return*** *\*/* **private static int** gcd2(**int** a, **int** b) {  
 **int** rem = 0;  
 **while** (b != 0) {  
 rem = a % b;  
 a = b;  
 b = rem;  
 }  
 **return** a;  
 }  
  
 @Test  
 **public void** method4() {  
 System.***out***.println(**"更相减损法：\n"** +  
 **" 第一步：任意给定两个正整数；判断它们是否都是偶数。若是，则用2约简；若不是则执行第二步。\n"** +  
 **" 第二步：以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。继续这个操作，直到所得的减数和差相等为止。\n"** +  
 **" 则第一步中约掉的若干个2与第二步中等数的乘积就是所求的最大公约数\n"** +  
 **" 其中所说的“等数”，就是最大公约数。求“等数”的办法是“更相减损”法。所以更相减损法也叫等值算法"**);  
 **int** a = 60, b = 24;  
 **int** quadrature = a \* b;  
 **int** gcd = 1;  
 **while** ((a & 1) == 0 && (b & 1) == 0) { *//如果a和b都是偶数* gcd = gcd << 1;  
 a = a >> 1;  
 b = b >> 1;  
 }  
 **while** (a != b) {  
 **if** (a > b) {  
 a -= b;  
 } **else** {  
 b -= a;  
 }  
 }  
 gcd \*= a;  
 System.***out***.println(**"最大公因数："** + gcd);  
 System.***out***.println(**"最小公倍数："** + (quadrature / gcd));  
 }  
  
}

1. 数组排序

private int[] arr = {2, 1, 3, 8, 1, 7, 0, 3, 5, 9};

冒泡排序BubbleSort

@Test

public void bubbleSort() {

System.out.println("冒泡排序，两两比较，把最大的依次放在最后，时间复杂度O(N^2)，额外空间复杂度O(1)");

int temp, len = arr.length;

//冒泡排序的优化：提前退出冒泡循环的标志位,即在这一次比较中没有交换任何元素，这个数组就已经是有序的了

boolean finish;

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

finish = true;

for (int j = 0; j < len - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

finish = false;

}

}

if (finish) break;

}

}

计算时间复杂度

O() = n - 1 + n - 2 + ... + 1 + 0 = (n - 1) \*n / 2 = O(n^2)

空间复杂度为O(1)

插入排序InsertionSort

@Test  
**public void** insertionSort2() {  
 System.***out***.println(**"插入排序，时间复杂度O(N^2)，额外空间复杂度O(1)。"**);  
 System.***out***.println(**" 插入排序（Insertion Sort）：最佳情况：T(n) = O(n) 最坏情况：T(n) = O(n2) 平均情况：T(n) = O(n2)\n"** +  
 **" 假定n是数组的长度，\n"** +  
 **" 首先假设第一个元素被放置在正确的位置上，这样仅需从1-n-1范围内对剩余元素进行排序。对于每次遍历，从0-i-1范围内的元素已经被排好序，\n"** +  
 **" 每次遍历的任务是：通过扫描前面已排序的子列表，将位置i处的元素定位到从0到i的子列表之内的正确的位置上。\n"** +  
 **" 将arr[i]复制为一个名为target的临时元素。\n"** +  
 **" 向下扫描列表，比较这个目标值target与arr[i-1]、arr[i-2]的大小，依次类推。\n"** +  
 **" 这个比较过程在小于或等于目标值的第一个元素(arr[j])处停止，或者在列表开始处停止（j=0）。\n"** +  
 **" 在arr[i]小于前面任何已排序元素时，后一个条件（j=0）为真，\n"** +  
 **" 因此，这个元素会占用新排序子列表的第一个位置。\n"** +  
 **" 在扫描期间，大于目标值target的每个元素都会向右滑动一个位置（arr[j]=arr[j-1]）。\n"** +  
 **" 一旦确定了正确位置j，\n"** +  
 **" 目标值target（即原始的arr[i]）就会被复制到这个位置。\n"** +  
 **" 与选择排序不同的是，插入排序将数据向右滑动，并且不会执行交换。"**);  
 **int** current, len = **arr**.**length**;  
 **for** (**int** i = 0; i < len - 1; i++) {  
 current = **arr**[i + 1];  
 **int** preIndex = i;  
 **while** (preIndex >= 0 && current < **arr**[preIndex]) {  
 **arr**[preIndex + 1] = **arr**[preIndex];  
 preIndex--;  
 }  
 **arr**[preIndex + 1] = current;  
 }  
 *//最佳情况下： 数组就是有序的，此时while不用循环了，时间复杂度就是最外层的for循环了，就是O(n)  
 //最坏清空下：数组是有序的，但是是降序排列的，我们目的是升序，  
 //比如： 6 5 4 3 2 1  
 //当i = 0 while遍历1次 数组是5 6 4 3 2 1  
 //当i = 1 while遍历2次 数组是4 5 6 3 2 1  
 //当i = 2 while遍历3次 数组是3 4 5 6 2 1  
 //......  
 //当i = n - 2 while遍历n-1次 数组就是升序了  
 // 1 + 2 + 3 + ... + n - 1 = (n - 1) \* n /2 = O(n^2)*}

选择排序SelectionSort

@Test  
**public void** selectionSort() {  
 System.***out***.println(**"选择排序，每次从现有数据挑出一个最小的直接放在最前面，第二次再将剩余的数据进行排序，拿出最小的放在最前面，以此类推"**);  
 **int** minIndex, temp, len = **arr**.**length**;  
 **for** (**int** i = 0; i < len - 1; i++) {  
 minIndex = i;  
 **for** (**int** j = i + 1; j < len; j++) {  
 **if** (**arr**[j] < **arr**[minIndex]) { *//找到最小的数* minIndex = j; *//将最小数的索引保存* }  
 }  
 **if** (minIndex != i) {  
 temp = **arr**[minIndex];  
 **arr**[minIndex] = **arr**[i];  
 **arr**[i] = temp;  
 }  
 *// 执行完一次循环，当前索引 i 处的值为最小值，直到循环结束即可完成排序* }  
 *//两个for循环无论如何都是要执行的，所以不存在什么最好和最坏的情况  
 //当i=0时，j=1到n-1需要遍历n-1次  
 //当i=1时，j=2到n-1需要遍历n-2次  
 //...  
 //当i=n-2时，j=n-1到n-1需要遍历0次  
 // n-1 + n-2 + ... + 1 + 0 = O(n^2)*}

希尔排序ShellSort

@Test  
**public void** shellSort() {  
 System.***out***.println(**"希尔排序，"**);  
 **int** len = **arr**.**length**, current, gap = len >>> 1;  
 **int** num = 0;  
 **while** (gap > 0) { *//第一个while A* System.***out***.println(++num);  
 *//当gap=1时，就是最简单的插入排序了* **for** (**int** i = gap; i < len; i++) { *// 第一个for B* current = **arr**[i];  
 **int** preIndex = i - gap;  
 **while** (preIndex >= 0 && **arr**[preIndex] > current) { *//第二个while C* **arr**[preIndex + gap] = **arr**[preIndex];  
 preIndex -= gap;  
 }  
 **arr**[preIndex + gap] = current;  
 }  
 gap = gap >>> 1;  
 }  
 *//当gap = n/2时，for必须执行（n-1-n/2 + 1）=n/2次 最好情况下while只执行一次就退出了，此时BC执行了n/2次，最坏情况下也只需要执行1次（2个数组为一组进行插入排序）  
 // 当i=n/2时 while执行1次  
 // 当i=n/2 + 1时 while执行1次  
 // 当i=n/2 + 2时 while执行1次  
 // ......  
 // 当i=n-1时 while执行1次  
 // 此时BC执行n/2次  
 //当gap = n/4时，for必须执行（n-1-n/4 + 1）=3n/4次 最好情况下while只执行一次就退出了，此时BC执行了3n/4次，  
 // 当i=n/4时 current=arr[n/4], preIndex = 0，preIndex依次是0，-n/4 while执行1次  
 // ..........  
 // 当i=2n/4时 current=arr[2n/4], preIndex = n/4，preIndex依次是n/4，0, -n/4 while执行2次  
 // ..........  
 // 当i=3n/4时 current=arr[3n/4], preIndex = 3n/4 - n/4= n/2，preIndex依次是n/2, n/4, 0, -n/4 while执行3次  
 // ..........  
 // 当i=n-1时 current=arr[n - 1], preIndex = n - 1 - n/4= 3n/4 - 1，preIndex依次是3n/4 - 1,2n/4 - 1,n/4 - 1,-1 while执行3次  
 // 此时BC执行n/4 + n/4 \* 2 + n/4 \* 3 = (1 + 2 + 3)\*n/4次  
 //当gap = n/8时，for必须执行（n-1-n/8 + 1）=7n/8次 最好情况下while只执行一次就退出了，此时BC执行了7n/8次，  
 // 此时BC执行(1 + 2 + ... + 7)\*n/8次  
 //......  
 //当gap=1时，是最后一次while循环了（完全等同于插入排序了），for循环必须执行(n-1-1+1)=n-1次，最好情况下while只执行一次就退出了，此时BC执行了n-1次  
 //最好情况下： n/2 +3n/4 + 7n/8 + ... + (n-1) =  
 //最坏情况下：n/2 + (1 + 2 + 3)\*n/4 + (1 + 2 + ... + 7)\*n/8 + ...... + (1 + 2 + ... + 2^())\*n/  
  
 //假设一共需要t次二分gap，也就是2^t <= n;*}

希尔排序时间复杂度计算公式

T(n) = ，其中

= 

< 

=

=



< 