|  |  |  |
| --- | --- | --- |
| 题号 | 提交次数 | 错误原因 |
| 1 | 2 | * map.containsKey 拼写错误 写成了map.containKeys |
| 2 | 3 | * 第一次：忘记了递增两个链表的迭代器 * 第二次：忘记处理最后一次进位的问题 |
| 3 | N | * right的更新，以及可以包含非字母的字符 |
| 4 | N | * 中位数的表达，第k顺序数从1开始 |
| 5 | N |  |
| 6 |  |  |
| 7 | 3 | * 没有处理溢出的情况 |
| 8 |  |  |
| 9 | 1 |  |
| 10 | 5 | dp：   * **没有处理模式串'.\*'时的初值** * 2、i重定义 * 3、漏写分号 |
| 10 | 3 | 递归：   * 注意点：递归是假定先前已匹配，否则不会执行当前次迭代 * 而dp不是这样，需要考虑前面是否匹配 |
| 11 | 5 | * 注意点：是根据迭代器的位置的高度来决定移动左边还是右边，而不是根据最左最右的边界来判断，因为迭代器的位置和左右边界的位置是分离的 |
| 12 | 2 | * "MC"->"CM" |
| 13 | 2 | * 迭代器递增错放在了使用之前 * 罗马数字是从长到短唯一可译，最长4位从4位开始匹配，直到1位为止 * **特殊情情况进行减法运算，其余进行加法运算** |
| 14 | 2 | 少些一个'）' |
| 15 | 3 | * for循环忘了递增 * 要求输出类型没看清 * 每一层加法都需要避免重复 |
| 16 | 3 | * 当寻找到比当前更接近的时候，迭代靠近的两指针不可跳过重复元素，‘接近于’和‘等于’不同，[1 2 2 2 2 3 3 3 3 ] 若果当1+2+3=6比无穷更接近5时，如果跳过所有重复元素就会丢失1+2+2这种可能性 |
| 17 | 3 | * 边界条件：当输入字符串为""时，返回空的List而不是含有""的List |
| 18 | 5 | * 迭代靠近的两指针，更新的时候写反了 |
| 19 | 1 |  |
| 20 | 2 | * 由于包含三种括号，**因此要用一个栈来记录上一个括号的类型，而不能只用一个char** |
| 21 | 1 |  |
| 22 | 3 | * **StringBuilder.delete(int start,int end)** * **回溯法中，表示状态的量都需要回溯！！！** |
| 23 | 1 |  |
| 24 | 3 | preLeft的更新要在left更新之前 |
| 25 | 3 | **注意点：长为k的链表的pre节点和tail节点，其中pre节点的更新节点为逆转前的k长链表真表头** |
| 26 | N | **呵呵** |
| 27 | 1 |  |
| 28 | 2 | 边界条件 |
| 29 |  |  |
| 30 |  |  |
| 31 |  |  |
| 32 | 2 |  |
| 33 | 3 | 题意理解错误 |
| 34 | 1 | * **非常好的二分法例子（注意判断条件以及左右端点的更新方式），思考如何避免讨论** * **注意 left+(right-left>>1)** * **注意初始条件，首先要让target落在nums[0]-nums[nums.length-1]的范围内，否则就不存在了** * **注意，哪个判断条件包含等号** * **比如if nums[mid]<=target** * **left=mid+1 //那么当while退出时，异常的边界是left，而right是所找的值** * **如果if nums[mid]>=target** * **right=mid-1 //当while退出时，异常的边界是right，而left是所找的值** |
| 35 | 1 | * **同理34，如何写出不用讨论的二分法解法** |
| 36 | 2 |  |
| 37 |  | **不会！**   * **如何根据row和col遍历其所在的cube）**   **i=row/3\*3;i<row/3\*3+3 j=col/3\*3;j<col/3\*3+3**   * **对于回溯法（需要判定是否可行的话，回溯函数得返回boolean，否则就是void）** |
| 38 | 1 |  |
| 39 | 1 |  |
| 40 | 2 | * **注意：需要排序后才能很好的避免重复** |
| 41 | 1 | **不会**   * **根据快排分组的思想，进行元素的划分，更新边界以及迭代器** |
| 42 | 1 | **不会**   * **左右两边，较低的一边进行更新** |
| 43 | N | * **将问题分解为：一个多位数与单个数的乘法与加法** |
| 44 | 1 | **\* ？匹配问题 有非常高效的方法！** |
| 45 |  | **不会**   * **Farthest的初始化以及更新位置是个问题** |
| 46 | 1 | * **需要增加一个状态存储数组used** |
| 47 | 2 | * **如何避免重复！！！（当前数字与前一个数字相同，且前一个数字没有被用时，才会跳过，如果前一个数字被用了，说明不是当前位置上的，不能算重复）** |
| 48 | 1 | * **一圈圈由外向内旋转：每一圈的旋转，遍历一条边上的点，一次交换对应4个点的顺序** |
| 49 | 1 | * **注意Map的遍历语法 *Map*.Entry Map.entrySet()** * ***Map*.Entry.getValue(),*Map*.Entry.getKey()** |
| 50 | 3 |  |
| 51 | N | * **搞清楚如何才算不冲突** |
| 52 | 1 |  |
| 53 | 1 |  |
| 54 | 2 | * **当前方向的末端点，在该方向上进行读取(iter<=right;...)，而非放到转换方向后进行读取(left<right;...)(只有一个数字会有无法读取的现象)** |
| 55 | 1 | **同45，需要初始化farthest；过了curEnd才进行跳跃** |
| 56 | 3 | * **注意pre的收尾** * **要对起始端点进行排序** * **Collections.sort(Collection<T>,Comparator<T>)用法** |
| 57 | 2 | * **不必先排序，56中的pre相当于新插入的区间** * **不重叠就加入，重叠就合并** |
| 58 | 2 |  |
| 59 | 1 |  |
| 60 |  | * **顺序数k是从1开始的，下标是从0开始的** * **用dp求出阶乘** |
| 61 | 2 | * **边界条件** |
| 62 | 2 | * **dp初始化条件，在循环中记得跳过（初始化下标为非额外下标）** |
| 63 | 1 |  |
| 64 | 1 |  |
| 65 |  |  |
| 66 | 2 | * **记得处理最后一位的进位** |
| 67 | 1 |  |
| 68 | 2 | * **没有处理一行中只有一个单词的情况** |
| 69 |  | * **不同讨论的二分法模式在这里失效了，为什么？** |
| 70 | 1 |  |
| 71 | 5 | * **不会** * **栈为空时遇到"/.."不用退出循环，可以继续往下走** |
| 72 | 2 | * **初始化处理** * **删除与插入是等价的** |
| 73 | 1 |  |
| 74 | 2 | * **二分法在这里也不太好用，需要判断** |
| 75 | 1 |  |
| 76 | 2 | * **while循环时，迭代器忘记递增了** |
| 77 | 2 | * **循环条件（由于begin是从1开始的）** |
| 78 | 2 | * **第i个位置（不放元素，或者放任意元素）** |
| 79 | 3 | * **漏写括号** * **没有判断是否已经用过了** |
| 80 | 3 | * **细节** |
| 81 | 1 |  |
| 82 | 2 |  |
| 83 | 1 |  |
| 84 |  | * **非常难** |
| 85 |  |  |
| 86 | 1 |  |
| 87 |  | * **不会** * **对s1的遍历只能到倒数第二个，必须保证左右两部分长度大于0，否则会无限循环** |
| 88 | 1 |  |
| 89 | 2 |  |
| 90 | 1 |  |
| 91 | N | * **dp：但需要两个数组存储** * **而且每个数组的初始化很重要** |
| 92 | 2 | * **给定pre节点以及next，反转中间的链表** * **初始化迭代器后：iter=pre.next** * **将pre与next进行连接：pre.next=tail（避免讨论）** |
| 93 | 2 | * **当第四组填完时，无论如何都要返回（在分情况是否是所需要的数据，不能把两个条件合在一起）** |
| 94 | 1 |  |
| 95 | N | * **由于要遍历左右孩子节点，根节点的创建时在循环中的，因此如果，左右孩子都不存在的话，需要额外添加一个null，否则都无法进入循环** * **或者，初始化边界条件的时候就添加null** |
| 96 | 1 |  |
| 97 |  | * **不会** |
| 98 | N and 1 | * **栈与递归法** * **递归：值域区间** |
| 99 | 1 | * **需要3个额外的空间，first，second，pre** |
| 100 | 1 | * **递归** |
| 101 | 1 |  |
| 102 | 1 |  |
| 103 | 1 |  |
| 104 | 1 |  |
| 105 | 1 | * **不会** * **前序遍历传入左端点索引** * **中序遍历传入左右两个索引** |
| 106 | 1 | * **同105** |
| 107 | 1 |  |
| 108 | 1 |  |
| 109 | 1 |  |
| 110 | 5 |  |
| 111 | 3 | * **注意与最大深度的区别** * **最大深度可以不用判断是否为叶节点** * **最小深度必须判断** |
| 112 | 2 |  |
| 113 | 2 |  |
| 114 |  | * **非常难理解的递归** |
| 115 |  |  |
| 116 | 1 |  |
| 117 | 1 |  |
| 118 | 2 |  |
| 119 | 2 |  |
| 120 | 3 |  |
| 121 | 5 |  |
| 122 | 1 |  |
| 123 | 5 | * **直接可以衍生到n次交易（当n小于days/2）否则就是122了** |
| 124 |  |  |
| 125 | 5 | * **注意！！！'A'-'a'是一个int型的数！！！** |
| 126 |  |  |
| 127 |  |  |
| 128 |  |  |
| 129 | 2 | * **将Integer的引用传入方法中，实际上是值传递，而且Integer的==运算符也是值比较，而非引用比较** |
| 130 |  | * **非常奇怪的错误，为什么>=0就会错误？见eclipse代码的注释** |
| 131 |  |  |
| 132 |  |  |
| 133 |  |  |
| 134 |  | * **需要两个量：油箱余量以及欠下的油量** |
| 135 |  | * **左右两遍遍历** |
| 136 | 1 |  |
| 137 | N | * **~~cnt[i]+=1<<i&num~~** * **cnt[i]=num>>i&1** |
| 138 | 3 | * **定义变量忘记注明类型** * **想不出更好的办法** |
| 139 |  |  |
| 140 |  |  |
| 141 | N |  |
| 142 | 2 |  |
| 143 | N |  |
| 144 | 1 |  |
| 145 | 3 | * **(通用法改一下)反向，从前插入，具体看leetcode2.Code145\_2** |
| 146 | 2 |  |
| 147 | N |  |
| 148 | N | * **Merge时，将尾节点置为null而不是原有的tail** |
| 149 |  |  |
| 150 | 3 | * **比较String内容时错用==** |
| 151 | N | * **String.split(" ")分割" 1 "会得到"" "1" ""三个String** |
| 152 | 1 | * **思考了很久，DP有点忘了** |
| 153 | 1 |  |
| 154 | 1 |  |
| 155 | 1 | * **又发现另一种实现方式** |
| 160 |  |  |
| 162 | 1 | * **我想的方法效率太低** |
| 164 | 1 | * **桶** |
| 165 | 2 |  |
| 166 | 2 |  |
| 167 |  | * **思考与第一题的区别** |
| 168 | N | * **与171联合，可以用反向思维，根据171的实现反推本题的解法** |
| 169 | 1 |  |
| 171 |  |  |
| 172 |  |  |
| 173 | 1 | * **中续遍历的堆栈法拆分成若干个函数而已** |
| 174 | 2 | * **注意dp关系式** |
| 179 | 3 | * **关键在于如何排序** |
| 187 | 2 | * **内容相同的String对象具有相同的hashCode** |
| 188 | 2 | * **dp关系式，很难解释** |
| 189 | 1 |  |
| 190 | 1 |  |
| 191 | 1 |  |
| 198 | 2 |  |
| 199 | 1 |  |
| 200 | 1 |  |
| 201 | 不会 | * **等价转换为相同的前缀** |
| 202 | 2 |  |
| 203 | 1 |  |
| 204 | 5 |  |
| 205 | 2 | * **注意初始的0** |
| 206 | 1 |  |
| 207 | 1 | * **BFS以及DFS** |
| 208 | 2 | * **boolean isWord是关键** |
| 209 | 5 | * **为什么第一反应是DP** |
| 210 | 1 | * **DFS有问题** |
| 211 |  |  |
| 212 |  |  |
| 213 | 3 | * **边界条件没有考虑** |
| 214 |  | * **非常巧妙的转化，KMP** |
| 215 |  | * **第k大，等价于第nums.length+1-k小** |
| 216 | 2 |  |
| 217 | 1 | * **居然用set呵呵** |
| 218 | 3 | * **记住这种方式吧** |
| 219 | 1 |  |
| 220 | N | * **t为0时，不要直接赋值为1，这样会导致在判断<=t的时候需要讨论了** |
| 221 | 1 | * **dp[row][col]=min(dp[row][col-1],dp[row-1][col],dp[row-1][col-1])+1** |
| 222 | N |  |
| 223 | 2 |  |
| 224 | N | * **'-'错写成了'\_' 碰到减号要把preVal=-curVal，而不是在sum更新的时候减** * **如果减号放在sum更新时，那么需要保留两个符号** |
| 225 | 1 |  |
| 226 | 1 |  |
| 227 | 1 | * **着重写一个带有括号的即可** |
| 228 | 1 |  |
| 229 | N | * **不太明白为什么是这样的条件分类** * **当count1==0时，为什么count2不用递减** |
| 230 | 1 |  |
| 231 | 2 | * **小心负数** |
| 232 | 1 | * **用了2个堆栈，在压入时或者弹出时调整即可** |
| 233 |  | * **这种题有啥意思？？？** |
| 234 | 1 | * **链表本身可以改造！** |
| 235 |  | * **只需从根要沿着往下走即可，分开那个节点就是** |
| 236 |  | * **我的方法是找出两个节点的路径，然后比较路径，效率比较低** |
| 237 | 1 |  |
| 238 |  | * **从左往右计算第i个数的从1-(i-1)的积** * **然后从右往左再来一次类似的过程** |
| 239 |  | * **用优先队列比较慢，用自定义的有序队列更快** |
| 240 |  | * **我错误的原因：我想要找到该target存在的某一行** |
| 241 |  | * **选择哪个符号最后运算，其实跟括号没啥关系的** |
| 242 | 1 |  |
| 257 | 3 | * **注意终止递归的条件不是当前节点为null而是，当前节点的子节点全是null** |
| 258 | 1 |  |
| 260 |  | * **根据一次异或的结果分成两拨** |
| 263 |  |  |
| 264 |  |  |
| 268 |  |  |
| 273 |  |  |
| 274 |  |  |
| 275 |  | * **二分法怎么思考** |
| 278 |  | * **二分法** |
| 279 |  |  |
| 282 |  |  |
| 283 | 2 |  |
| 284 |  |  |
| 287 |  |  |
| 289 |  | * **状态转移，既标志将去的状态又保留之前的状态** |

|  |  |
| --- | --- |
| 数组相关 | 7、9、11、26、27、31、36、41、42、48、54、56、57、59、60、73、75、76、80、81、84、85、88、89、118、119、120、121、122、123、136、137、152、153、154、162、169、189、190、191、200、205、209、217、219、228、229、238、239、240、242、258、260、268、274、275、283、287、289 |
| 算数 | 1、2、15、16、18、43、66、67、69、166、167、168、171、172、190、191、201、202、204、223、231、263、264 |
| 回溯法（子函数返回类型void） | 17、22、39、40、46、47、51、52、77、78、90、93 |
| 回溯法（子函数返回类型boolean） | 37、79 |
| 动态规划 | 3、5、10、44、32、38、53、62、63、64、70、72、91、95、96、115、120、123、152、174、188、198、213、221、241 |
| 贪心算法 | 45、55 |
| 堆栈 | 144、145、150、173、226、230 |
| 队列 | 102、103、104、107、199、218、239 |
| 递归 | 10、44、50、87、94、98、99、100、101、102、103、104、105、106、107、108、109、110、111、112、113、114、116、117、144、145、216、222 |
| 链表 | 21、23、24、25、61、81、82、86、92、109、138、141、142、143、147、148、160、203、206、234、237 |
| 树 | 94、95、96、98、99、100、101、102、103、104、105、106、107、108、109、110、111、112、113、114、116、117、144、145、199、222、226、230、235、236、257 |
| 桶 | 164、220 |
| 二分法 | 4、33、34、35、74、275、278 |
| 映射 | 12、13、 |
| 字符串 | 10、14、28、44、49、58、68、71、76、87、115、125、151、165、179、187、205、214、273 |
| 括号 | 20、22、32 |
| 数据结构的实现 | 146、155、208、211、224、225、227、232、284 |
| 图 | 207、210 |
| 基本算法 | 215 |

DP初始化：

* 额外的下标（比如字符串匹配），那么在循环时不用判断跳过初始的情况
* 非额外的下标（比如62），那么在循环时需要判断，并且跳过初始条件，否则就会重新写入初始化的条件了。