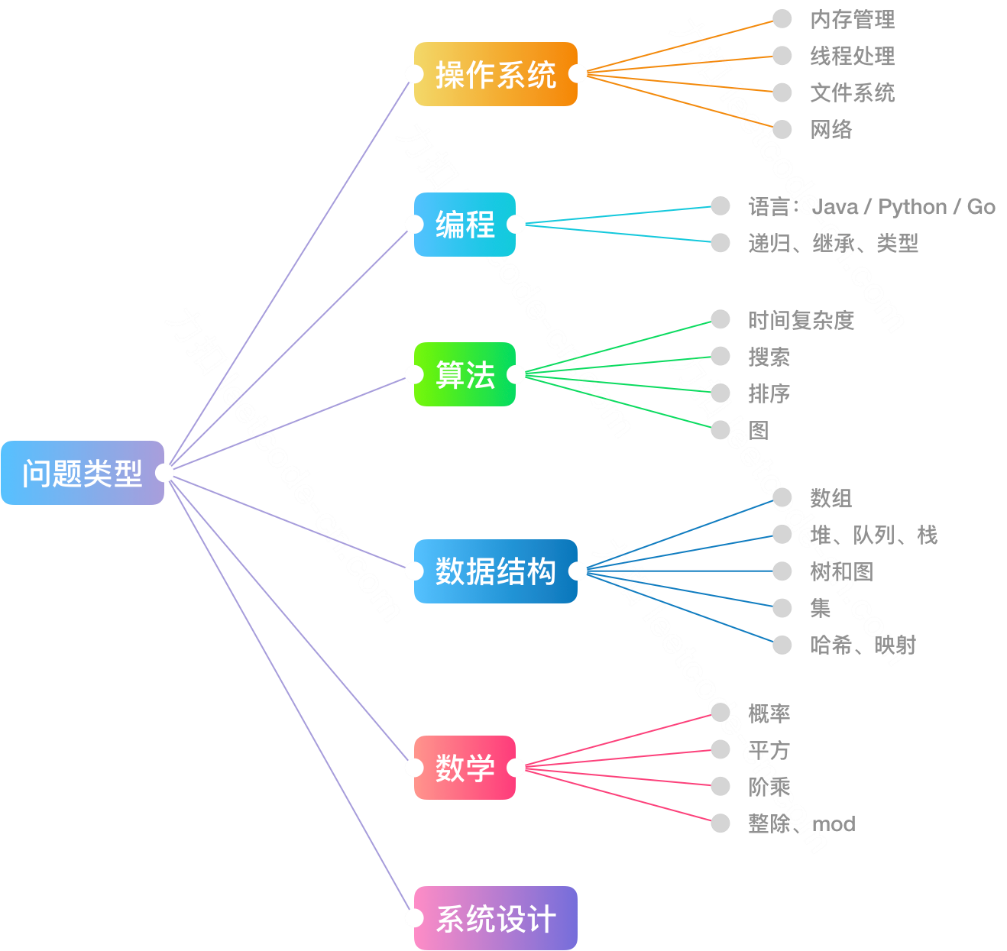
* 数据结构基本概念
  + 基本概念和术语
    - 数据：对客观事物的符号表示
    - 数据元素：数据的基本单位
    - 数据对象：具有相同性质的数据元素的集合
  + 数据结构：是计算机存储、组织数据的方式，是指相互之间存在一种或多种特定关系的数据元素的集合
    - 逻辑结构
      * 集合结构：元素之间的关系仅为处于同一集合
      * 线性结构：元素之间是一对一关系
      * 树形结构：元素之间是一对多关系
      * 图形结构：元素之间是多对多关系
    - 物理结构
      * 顺序结构：元素的地址是连续的
      * 链式结构：元素的地址是不连续的
      * 索引存储：存储元素信息的同时，还建立附加的索引表，索引表中的每项称为索引项，一般形式为（关键字，地址）
      * 散列存储：根据元素的关键字直接计算出元素地址
    - 操作：对数据元素的运算（增删改查）
* 算法和算法性能度量
  + 算法：特定问题求解步骤的一种描述，若干条指令组成的有穷序列，每条指令表示一个或多个操作
    - 输入：\*个输入
    - 输出：+个输出
    - 有穷性：必须总是在执行有穷步后结束，且每一步都可在有穷时间内完成
    - 确定性：每条指令含义必须明确，无二义性——即**相同的输入只能得到相同的输出**
    - 可行性：算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现
  + 算法性能度量
    - 时间复杂度：设问题规模为n，基本操作重复执行的次数是n的某个函数T(n)，若存在某个辅助函数，使得n趋于∞时T(n)/f(n)为不等于0的常数，记作T(n)=O(f(n)，称O(f(n))为算法的渐进时间复杂度，简称时间复杂度
    - 空间复杂度：S(n)=O(f(n))
    - 常见复杂度
      * O(1)
      * O(log n)
      * O(n)
      * O(n log n)
      * O(n^k)
      * O(k^n)
      * O(n!)
      * O(n^n)
  + 程序=算法+数据结构
* 线性表
  + 定义：n个**类型相同**的数据元素组成的**有限序列**
  + 特点
    - **有且仅有一个**开始节点和终端节点
    - 开始节点无直接前趋，**有且仅有一个**直接后继；终端节点无直接后继，**有且仅有一个**直接前趋；其余任何节点**有且仅有一个**直接前趋和后继
  + 运算：置空，求长度，根据索引获取值，根据值获取索引，插入，删除
  + 不同存储结构表示
    - 顺序存储：**顺序表**，用一组地址连续的存储单元存储线性表的元素；数据元素逻辑上相邻，物理上也相邻
    - 链式存储：**链表**，不要求逻辑上相邻的元素物理上也相邻
      * 单链表：每个节点存放元素自身信息外，还存放一个指向其后继的指针
      * 双链表：存放指向前趋和后继的指针
      * 循环链表
        + 循环单链表：尾节点存放指向头节点的指针
        + 循环双链表：头尾节点存放指向
      * 静态链表：借助数组描述链表，有数据域和指针域，但指针域存放节点相对地址即**数组下标——游标**
    - 顺序表和链表比较
      * 存取方式
        + 顺序表可以顺序存取也可以随机存取
        + 链表只能从表头顺序存取元素
      * 逻辑和物理结构
        + 顺序表的元素逻辑物理上都相邻
        + 链表的元素物理上不一定相邻
      * 查找、插入和删除操作
        + 无序按值查找，两者复杂度都是O(n)，有序时顺序表可以用二分查找；按序号查找，顺序表常数复杂度，链表线性复杂度
        + 插入删除，顺序表平均复杂度为线性复杂度，链表为常数复杂度
      * 空间分配
        + 顺序表静态分配效率高，容量固定；动态分配效率低，可以扩容
        + 链表分配灵活，效率高
* 栈和队列
  + 栈：只允许在**同一端**进行插入和删除操作的**线性表**
    - 顺序栈
    - 链栈
  + 队列：只允许在表的**一端进行插入**，而在**另一端进行删除**的**线性表**
    - 顺序存储
      * 顺序队列
      * 循环队列
    - 链式队列
    - 双端队列：允许两端都可以进行入队和出队操作的队列
      * 输出端受限：一端可以插入删除，另一端只能插入
      * 输入端受限：一段可以插入删除，另一端只能删除
* 树与二叉树
  + 树定义：n个节点的有限集合，非空树应满足
    - 有且只有一个根节点
    - n>1时，其余节点可以分为m个互不相交的有限集合，每个集合本身又是一棵树，称为根节点的子树
    - 根节点没有前趋节点，其他所有节点有且只有一个前趋
    - 所有节点可以有\*个后继节点
* 串
  + 定义：0或多个字符组成的有限序列
* 数组和广义表
  + 数组存储方式：一般采用顺序存储
  + 特殊矩阵
    - 对称矩阵
    - 三角矩阵
  + 稀疏矩阵



* 排序算法知道哪些？冒泡排序的时间复杂度和空间复杂度？

1. 排序算法种类
   1. 冒泡：两两比较，将大的元素放后面
   2. 选择：每次从无序的部分**选择**出最小的元素和**无序部分**的第一个元素的位置交换
   3. 插入：对于未排序数据，在已排序序列中**从后向前**扫描，找到相应的位置并插入，保证大于左边小于右边
   4. 归并：把待排序序列分为若干个子序列（分治法），每个子序列是有序的。然后再把有序子序列合并为整体有序序列
   5. 快排：任取一个分界值，大于划分值放在右边，小于划分值放在左边，然后分别递归处理划分值的左右两边
   6. 希尔：将待排序的数组元素分成多个子序列，使得每个子序列的元素个数相对较少，然后对各个子序列分别进行直接插入排序，待整个待排序列“基本有序”后，最后在对所有元素进行一次直接插入排序
   7. 堆排序：使用堆这种数据结构进行排序
   8. 基数排序
   9. Bitmap：把计数排序用的统计数组的每个单位缩小成bit级别的布尔数组

2. 时间空间复杂度



* 数组：查找修改效率高；增删效率低
  + 在一个1到100的整数数组中找到丢失的数字
    - 丢失1个：1-100求和 – 所有元素求和即可
    - 丢失多个：排序
  + 在给定的整数数组中找到重复的数字
    - 排序
    - 用HashSet
  + 在未排序整数数组中找到最大值和最小值：遍历并更新最大最小值
  + 找到数组所有和等于一个给定数的数对：使用HashSet查找val和sum-val是否同时存在
  + 查找数组或字符串中（第一个）不重复元素：使用LinkedHashMap或者Set+List(遍历，Set放重复的值，List放不重复的值)
  + Java中如何从给定数组中删除多重复值：排序
  + 使用快速排序算法对整数数组进行排序

private void quickSort(int low, int high) {

int i = low; int j = high; // pivot is middle index

int pivot = input[low + (high - low) / 2]; // Divide into two arrays

while (i <= j) {

while (input[i] < pivot) i++;

while (input[j] > pivot) j--;

if (i <= j) {

swap(i, j); // move index to next position on both sides

i++; j--;

}

}

if (low < j) quickSort(low, j);

if (i < high) quickSort(i, high);

}

* + 倒置数组：先转换为list，用Collections.reverse(list)，再转换为数组
  + 获得出现次数超过长度一半的数：摩尔投票法，维护一个计数器和保存出现超过一半长度次数的变量，遍历数组中的元素，若计数器为0则缓存为当前元素，若计数器不为0，当前元素=缓存元素则计数器递增；否则递减，最终一次遍历后得到的缓存元素即为所求结果<https://www.nowcoder.com/practice/e8a1b01a2df14cb2b228b30ee6a92163?tpId=13&tqId=11181&tPage=2&rp=1&ru=%2Fta%2Fcoding-interviews&qru=%2Fta%2Fcoding-interviews%2Fquestion-ranking>
  + 获取拼接在一起的字符串代表的数最大/小的结果：双层循环的上三角遍历方式，每次用字符串的compareTo方法比较两者前后顺序不同时拼接的结果，若颠倒后结果大于原先顺序交换内外层指向的字符串<https://leetcode-cn.com/explore/interview/card/top-interview-questions/270/sort-search/1169/>
* 链表：增删效率高；查找修改效率低
  + 在一次遍历中找到单个链表的中间元素：双指针法，快指针每次指向下一个，慢指针每两次指向下一个，快指针遍历到结尾时慢指针所指元素即为中间元素
  + 证明给定的链表是否包含循环：快慢指针，快指针一次走两步，慢指针一次走一步，两者相等时存在环，快指针为空时不存在
  + 找到循环的头节点：HashMap
  + 反转链表：若头节点为空或头节点的下一个节点为空则返回头节点
    - 迭代

ListNode cur = head;

while(cur != null && cur.next != null){

ListNode newHead = cur.next;

cur.next = newHead.next;

newHead.next = head;

head = newHead;

}

return head;

* + - 递归

ListNode p = reverseList(head.next);

head.next.next = head;

head.next = null;

return p;

* + 删除未排序链表中的重复节点：嵌套循环
  + 得到单链表的长度：遍历
  + 获取单链表中倒数第n个节点：快慢指针，计数器从0开始，快指针每次前进一步，计数器+1，当计数器的值>n时慢指针前进一步，直到快指针到**尾节点**

if(head == null || k == 0)

return null;

ListNode fast = head;

ListNode slow = head;

int start = 1;

while (fast.next != null) {

fast = fast.next;

start++;

if (start > k) slow = slow.next;

}

if(k > start)

return null;

return slow;

* + 合并排序链表：用队列
  + 排序链表：归并排序<https://leetcode-cn.com/problems/sort-list/>
* 字符串
  + 找到字符串中的重复字符
    - 关心重复次数：Map
    - 不关心次数：Set
  + 两个字符串是否相互异构：Map
  + 打印第一个非重复字符：LinkedHashMap
  + 反转字符串
* if (str.length() < 2) {

return str;

}

return reverseRecursively(str.substring(1)) + str.charAt(0);

* + 双指针，相互交换，直到前指针越过后指针<https://leetcode-cn.com/explore/interview/card/top-interview-questions/275/string/1144/>
* 计算字符串中特定字符出现次数：简单：switch；复杂：map
* 判断字符串是否只包含数字：s.matches("^[-+]?[0-9]\*(?:\\.[0-9]\*)?(?:[eE][-+]?[0-9]+)?$")
* 找到字符串的所有排列：回溯法<https://leetcode-cn.com/problems/zi-fu-chuan-de-pai-lie-lcof/submissions/>
* 不使用任何库方法的情况下逆转一个句子中的单词：string.split+StringBuilder
* 判断两个字符串是否相互旋转：首先比较两个字符串长度，相等则将其中一个字符串拼接在自身尾部成为一个长字符串，用indexOf判断另一个字符串是否为长字符串的子串
* 检查给定的字符串是否回文：双指针
* Double转二进制字符串：<https://leetcode-cn.com/problems/bianry-number-to-string-lcci/submissions/>
* 二叉树
  + 实现二叉搜索树：
  + 遍历
    - 前序：栈，<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/2/traverse-a-tree/1/>
    - 中序：栈，<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/2/traverse-a-tree/2/>
    - 后序：栈，<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/2/traverse-a-tree/3/>，前序遍历改变访问左右子树顺序然后反转
    - 层序：队列，<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/2/traverse-a-tree/9/>
  + 访问所有叶子节点：若节点为空则返回，若左右子树为空则为叶子并访问，否则对其左右子树递归调用
  + 计算叶节点数：若节点为空则返回0，若左右子树为空则为叶子并返回1，否则返回左右子树递归调用结果的和
  + 二分搜索：<https://leetcode-cn.com/explore/learn/card/binary-search/209/template-i/835/>
  + 最大深度：<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/3/solve-problems-recursively/12/>
  + 对称二叉树：<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/3/solve-problems-recursively/13/>
  + 平衡二叉树：<https://leetcode-cn.com/problems/balanced-binary-tree/submissions/>
  + 是否存在路径和等于某个数：<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/3/solve-problems-recursively/14/>
  + 最近公共祖先：<https://leetcode-cn.com/explore/learn/card/data-structure-binary-tree/4/conclusion/19/>
  + 序列化和反序列化：<https://www.nowcoder.com/practice/cf7e25aa97c04cc1a68c8f040e71fb84?tpId=13&tqId=11214&tPage=4&rp=2&ru=%2Fta%2Fcoding-interviews&qru=%2Fta%2Fcoding-interviews%2Fquestion-ranking>
  + 二叉搜索树中第k小元素：中序遍历，每次弹栈计数器加一直到和k相等<https://leetcode-cn.com/explore/interview/card/top-interview-questions/269/tree/1165/>
  + 获取指定二叉树节点的中序遍历的下一个节点：<https://www.nowcoder.com/practice/9023a0c988684a53960365b889ceaf5e?tpId=13&tqId=11210&tPage=3&rp=1&ru=%2Fta%2Fcoding-interviews&qru=%2Fta%2Fcoding-interviews%2Fquestion-ranking>
  + 用两个序列重建二叉树
    - 前序和中序
    - 后序和中序
  + 判断一棵树是否是另一棵树子结构：<https://www.nowcoder.com/practice/6e196c44c7004d15b1610b9afca8bd88?tpId=13&tqId=11170&tPage=1&rp=1&ru=%2Fta%2Fcoding-interviews&qru=%2Fta%2Fcoding-interviews%2Fquestion-ranking>
* 栈
* 队列
* 其他问题
  + 不使用临时变量交换两个整型值
    - 加法：a = a+b; b = a – b; a = a – b;
    - 异或：a = a^b; b = a ^ b; a = a ^ b;
  + 不使用加减乘除实现加法：return b == 0？a ：add((a^b), (int)(a &b) << 1)
  + 二进制中1的个数：<https://www.nowcoder.com/practice/8ee967e43c2c4ec193b040ea7fbb10b8?tpId=13&tqId=11164&tPage=1&rp=1&ru=/ta/coding-interviews&qru=/ta/coding-interviews/question-ranking>
  + 二分搜索
    - 求平方根：<https://leetcode-cn.com/explore/learn/card/binary-search/209/template-i/836/>
  + 回溯法
    - N皇后问题
    - 全排列：<https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-leetcode-solution-2/>
  + 动态规划
    - 丑数：某个丑数=较小丑数\*2/3/5<https://leetcode-cn.com/problems/chou-shu-lcof/>
    - 最大子序和：<https://leetcode-cn.com/problems/maximum-subarray/>
    - 斐波那契数列、爬楼梯
    - 打家劫舍：<https://leetcode-cn.com/problems/house-robber/>
    - 完全平方数之和：<https://leetcode-cn.com/problems/perfect-squares/>
  + 判断矩形覆盖

public boolean isOverLapping(Rectangle other)

{

 if (this.topLeft.x > other.bottomRight.x

|| this.bottomRight.x < other.topLeft.x

|| this.topLeft.y < other.bottomRight

|| this.bottomRight.y > other.topLeft.y) return false;

 return true;

}