- 数据结构
  - 数组
    - 用一组连续的内存空间，存储一组相同类型的数据
    - CPU缓存可以预读，提高性能
    - 缺点是大小固定，如果分配的空间不够，需要再申请一段新的内存空间，把原数据拷贝过去，非常费时
    - 复杂度
      - 插入/删除/查找O(n)
      - 根据下标查找O(1)
    - 题目
      - https://leetcode-cn.com/problems/container-with-most-water/
      - https://leetcode-cn.com/problems/move-zeroes/
      - https://leetcode-cn.com/problems/climbing-stairs/
      - https://leetcode-cn.com/problems/3sum/
      - https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array/
      - https://leetcode-cn.com/problems/rotate-array/
      - https://leetcode-cn.com/problems/merge-two-sorted-lists/
      - https://leetcode-cn.com/problems/merge-sorted-array/
      - https://leetcode-cn.com/problems/two-sum/
      - https://leetcode-cn.com/problems/move-zeroes/
      - https://leetcode-cn.com/problems/plus-one/
  - 链表
    - 普通链表
      - 单向链表
      - 双向链表
        - 双向链表可以支持 O(1) 时间复杂度的情况下找到前驱结点
      - 循环链表
        - 循环链表的优点是从链尾到链头比较方便
      - 双向循环链表
      - 复杂度
        - 插入/删除O(1)
          - 实际是O(n)，因为先要查找目标
        - 查找O(n)
    - 链表的数据不是连续的，所以可以动态扩容
    - 应用场景
      - LRUCache
    - 跳表
      - 只能用于元素有序的情况
      - 增加了多级索引
      - 对标平衡树和二分查找
      - 复杂度
        - 插入/删除/查找O(log n)
        - 空间复杂度是O(n)
    - 题目
      - https://leetcode-cn.com/problems/reverse-linked-list/
      - https://leetcode-cn.com/problems/swap-nodes-in-pairs
      - https://leetcode-cn.com/problems/linked-list-cycle
      - https://leetcode-cn.com/problems/linked-list-cycle-ii
      - https://leetcode-cn.com/problems/reverse-nodes-in-k-group/
      - https://leetcode-cn.com/problems/merge-k-sorted-lists/
  - 栈
    - 先入后出
    - 复杂度
      - 插入/删除O(1)
    - 题目
      - https://leetcode-cn.com/problems/valid-parentheses/
      - https://leetcode-cn.com/problems/min-stack/
      - https://leetcode-cn.com/problems/largest-rectangle-in-histogram
      - https://leetcode-cn.com/problems/trapping-rain-water/
  - 队列
    - 先入先出
    - 普通队列
      - 复杂度
        - 插入/删除O(1)
        - 查找O(n)
        - 查找O(n)
    - 双端队列
      - 两端可以进出的 Queue
      - 复杂度
        - 插入/删除O(1)
        - 查找O(n)
      - Python中的双端队列
        - 基于双向链表实现
    - 优先队列
      - 复杂度
        - 插入O(1)
        - 取出O(log n)
          - 按照元素的优先级取出
        - 查找O(n)
    - 题目
      - https://leetcode-cn.com/problems/sliding-window-maximum
      - https://leetcode-cn.com/problems/design-circular-deque
  - 哈希表
    - 复杂度
      - 插入/删除/查找O(1)
    - 题目
      - https://leetcode-cn.com/problems/valid-anagram/
      - https://leetcode-cn.com/problems/group-anagrams/
      - https://leetcode-cn.com/problems/two-sum/
  - 树
    - 链表是特殊化的树
    - 树是特殊化的图
    - 二叉树
      - 遍历
        - 就是递归
        - 前序
          - 根左右
        - 中序
          - 左根右
        - 后序
          - 左右根
      - 完全二叉树
        - 除了叶子结点外都是满的
        - 找左子结点
          - 2i+1
        - 找右子结点
          - 2i+2
        - 找父结点
          - (i-1)/2
    - 二叉搜索树
      - 有序二叉树
      - 左子树上所有结点的值均小于它的根结点的值
      - 右子树上所有结点的值均大于它的根结点的值
      - 中序遍历就是升序排列
      - 复杂度
        - 查询/插入/删除O(log n)
      - 删除根结点
        - 找到第一个大于它的结点替换
    - 题目
      - 1. https://leetcode-cn.com/problems/binary-tree-inorder-traversal/
      - https://leetcode-cn.com/problems/binary-tree-preorder-traversal/
      - https://leetcode-cn.com/problems/n-ary-tree-postorder-traversal/
      - https://leetcode-cn.com/problems/n-ary-tree-preorder-traversal/
      - https://leetcode-cn.com/problems/n-ary-tree-level-order-traversal/
  - 堆
    - 可以迅速找到一堆数中最大值或最小值的数据结构
      - 根结点是最大值或最小值
    - 实现方式
      - 二叉堆
        - 通过完全二叉树实现
        - 是堆的一种常见且简单的实现，但不是最优实现
        - 树中任意结点的值总是大于其子结点的值（大顶堆）
        - 复杂度
          - 大顶堆
            - 找最大值O(1)
            - 删最大值O(log n)
            - 插入O(log n) or O(1)
          - 小顶堆
            - 找最小值O(1)
            - 删最小值O(log n)
            - 插入O(log n) or O(1)
      - 斐波那契堆
        - 时间空间复杂度更好
      - 工程中一般使用优先队列
    - 插入操作
      - 先插到堆的尾部
      - 依次向上调整整个堆的结构
        - heapifyUp
    - 删除最大元素
      - 把堆尾元素替换到顶部
      - 依次向下调整整个堆的结构
        - heapifyDown
      - 向下替换时，跟左右结点中较大的一个进行替换
    - 题目
      - https://leetcode-cn.com/problems/zui-xiao-de-kge-shu-lcof/
      - https://leetcode-cn.com/problems/sliding-window-maximum/
      - https://leetcode-cn.com/problems/top-k-frequent-elements/
      - https://leetcode-cn.com/problems/chou-shu-lcof/
  - 图
    - 点
      - 度
        - 入度
        - 出度
      - 是否连通
    - 边
      - 向
        - 有向
        - 无向
      - 权重
    - 分类
      - 无向无权
      - 无向有权
      - 有向无权
      - 有向有权
    - 题目
      - https://leetcode-cn.com/problems/number-of-islands/

- 算法
  - 递归
    - 题目
      - https://leetcode-cn.com/problems/climbing-stairs/
      - https://leetcode-cn.com/problems/generate-parentheses/
      - https://leetcode-cn.com/problems/invert-binary-tree/
      - https://leetcode-cn.com/problems/validate-binary-search-tree/
      - https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/
      - https://leetcode-cn.com/problems/minimum-depth-of-binary-tree/
      - https://leetcode-cn.com/problems/serialize-and-deserialize-binary-tree/
      - https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree/
      - https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/
      - https://leetcode-cn.com/problems/combinations/
      - https://leetcode-cn.com/problems/permutations/
      - https://leetcode-cn.com/problems/permutations-ii/
  - 分治、回溯
    - 题目
      - https://leetcode-cn.com/problems/majority-element/
      - https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/
      - https://leetcode-cn.com/problems/n-queens/
      - https://leetcode-cn.com/problems/powx-n/
      - https://leetcode-cn.com/problems/subsets/
  - 搜索
    - 深度优先搜索（DFS）
      - 递归实现
      - 栈实现
    - 广度优先搜索（BFS）
      - 队列实现
    - 题目
      - https://leetcode-cn.com/problems/binary-tree-level-order-traversal/
      - https://leetcode-cn.com/problems/minimum-genetic-mutation/
      - https://leetcode-cn.com/problems/generate-parentheses/
      - https://leetcode-cn.com/problems/find-largest-value-in-each-tree-row/
      - https://leetcode-cn.com/problems/word-ladder/
      - https://leetcode-cn.com/problems/word-ladder-ii/
      - https://leetcode-cn.com/problems/number-of-islands/
      - https://leetcode-cn.com/problems/minesweeper/
  - 贪心算法
    - 在局部做最优选择，从而希望在全局获得最优结果
    - 贪心法一般不能得到我们想要的答案，但是一个问题如果可以用贪心法解决，那么贪心法一般是这个问题的最优解
    - 与动态规划的区别
      - 贪心算法对每个子问题的选择不能回退
      - 动态规划可以保存以前的运算结果，并根据以前的结果对当前进行选择，有回退功能
    - 适用场景
      - 子问题最优解能递推到最终问题的最优解
      - 这种子问题最优解称为最优子结构
    - 题目
      - https://leetcode-cn.com/problems/coin-change/
      - https://leetcode-cn.com/problems/lemonade-change/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/
      - https://leetcode-cn.com/problems/assign-cookies/
      - https://leetcode-cn.com/problems/walking-robot-simulation/
      - https://leetcode-cn.com/problems/jump-game/
      - https://leetcode-cn.com/problems/jump-game-ii/
  - 二分查找
    - 前提
      - 目标函数单调性
      - 存在上下界
      - 能够通过索引访问
    - 题目
      - https://leetcode-cn.com/problems/sqrtx/
      - https://leetcode.com-cn/problems/valid-perfect-square/
      - https://leetcode-cn.com/problems/search-in-rotated-sorted-array/
      - https://leetcode-cn.com/problems/search-a-2d-matrix/
      - https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array/
  - 动态规划
    - 特点
      - 找到子问题，通过解决子问题，得出原问题的解
      - 会存储子问题的解
        - 当有重复子问题时，直接获取子问题的解
      - 只能应用于有最优子结构的问题
        - 中途淘汰次优解
    - 关键点
      - 最优子结构
      - 存储中间状态
      - 递推公式（状态转移方程、DP方程）
    - 题目
      - https://leetcode-cn.com/problems/climbing-stairs/
      - https://leetcode-cn.com/problems/triangle/
      - https://leetcode-cn.com/problems/maximum-subarray/
      - https://leetcode-cn.com/problems/maximum-product-subarray/
      - https://leetcode-cn.com/problems/coin-change/
      - https://leetcode-cn.com/problems/house-robber/
      - https://leetcode-cn.com/problems/house-robber-ii/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iii/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iv/
      - https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/
      - https://leetcode-cn.com/problems/perfect-squares/
      - https://leetcode-cn.com/problems/edit-distance/ （重点）
      - https://leetcode-cn.com/problems/jump-game/
      - https://leetcode-cn.com/problems/jump-game-ii/
      - https://leetcode-cn.com/problems/unique-paths/
      - https://leetcode-cn.com/problems/unique-paths-ii/
      - https://leetcode-cn.com/problems/unique-paths-iii/
      - https://leetcode-cn.com/problems/coin-change/
      - https://leetcode-cn.com/problems/coin-change-2/
      - https://leetcode-cn.com/problems/longest-valid-parentheses/
      - https://leetcode-cn.com/problems/minimum-path-sum/
      - https://leetcode-cn.com/problems/decode-ways/
      - https://leetcode-cn.com/problems/maximal-square/
      - https://leetcode-cn.com/problems/max-sum-of-rectangle-no-larger-than-k/
      - https://leetcode-cn.com/problems/frog-jump/
      - https://leetcode-cn.com/problems/split-array-largest-sum/
      - https://leetcode-cn.com/problems/student-attendance-record-ii/
      - https://leetcode-cn.com/problems/task-scheduler/
      - https://leetcode-cn.com/problems/palindromic-substrings/
      - https://leetcode-cn.com/problems/minimum-window-substring/
      - https://leetcode-cn.com/problems/burst-balloons/