数据结构与算法

数据结构

数组与链表

- 数组: 一段连续的内存空间, 通过下标访问元素, 时间复杂度O(1), 插入和删除元素的时间复杂度O(n)
- 链表: 由节点组成, 每个节点包含数据和指向下一个节点的指针, 时间复杂度O(n), 插入和删除元素的时间复杂度O(1)

栈与队列

- 栈: 先进后出, 只能在栈顶操作, 用于函数调用, 表达式求值等
- 队列: 先进先出, 只能在队头和队尾操作, 用于广度优先搜索等

哈希表

- 哈希表: 通过哈希函数将关键字映射到哈希表中的位置, 时间复杂度O(1), 用于查找, 插入和删除元素

树

- 二叉树: 每个节点最多有两个子节点, 用于查找, 排序等
    - 二叉搜索树: 左子树的所有节点的值小于根节点的值, 右子树的所有节点的值大于根节点的值, 用于查找, 排序等
    - 平衡二叉树: 左右子树的高度差不超过1, 用于查找, 排序等(是一种特殊的二叉搜索树)
    - 完全二叉树: 除了最后一层, 其他层的节点数都是满的, 用于堆等
- 红黑树: 一种自平衡二叉搜索树, 用于查找, 插入和删除元素
    - 特点: 根节点是黑色, 每个叶子节点是黑色, 每个红色节点的两个子节点都是黑色, 任意节点到叶子节点的路径包含相同数量的黑色节点
    - 时间复杂度: O(logn)
    - 用途: STL中的map和set就是基于红黑树实现的
    - 实现: 插入和删除元素时, 需要通过旋转和变色来保持平衡

堆

- 堆: 一种特殊的树形数据结构, 用于查找最大值或最小值
    - 大顶堆: 父节点的值大于等于子节点的值
    - 小顶堆: 父节点的值小于等于子节点的值

图

- 图: 由节点和边组成, 用于表示网络结构, 用于查找, 最短路径等
    - 有向图: 边有方向
    - 无向图: 边没有方向
    - 加权图: 边有权重

算法

1. 搜索算法

二分查找

- 二分查找: 通过比较中间元素和目标元素的大小, 缩小查找范围, 时间复杂度O(logn), 用于有序数组的查找

深度优先搜索 DFS

一般用于树和图的遍历

- 深度优先搜索: 从起始节点开始, 递归访问相邻节点, 直到没有未访问的节点
- 时间复杂度: O(V+E), V是节点数, E是边数
- 用途: 解决连通性问题, 拓扑排序等

深度优先搜索的实现步骤：

1. **初始化**：使用递归或栈来进行遍历，同时使用一个数组或集合来记录访问的节点。

2. **从起点开始**：将起始节点标记为已访问，并递归访问或使用栈探索其未访问的邻居。
3. **递归进行**：对于每个节点的每一个未访问的邻居，重复执行 DFS 过程，直到所有节点都被访问。

广度优先搜索 BFS

一般用于树和图的遍历

- 广度优先搜索: 从起始节点开始, 逐层访问相邻节点, 直到找到目标节点
- 时间复杂度: O(V+E), V是节点数, E是边数
- 用途: 解决最短路径问题, 连通性问题等

广度优先搜索的实现步骤：

1. **初始化**：使用一个队列来存储将要访问的节点，使用一个数组或集合来标记已访问的节点。
2. **从起点开始**：将起始节点加入队列，并标记为已访问。
3. **逐层遍历**：
    - 取出队列的第一个节点，访问该节点的所有未访问的邻居节点。
    - 将所有未访问的邻居加入队列并标记为已访问。
4. **重复**：重复上述步骤，直到队列为空为止。

2. 排序算法

选择排序

- 选择排序: 每次选择最小的元素, 放到已排序的末尾
- 时间复杂度: O(n^2), 空间复杂度: O(1), 稳定性: 不稳定

冒泡排序

- 冒泡排序: 两两比较相邻元素, 如果顺序错误, 则交换位置
- 时间复杂度: O(n^2), 空间复杂度: O(1), 稳定性: 稳定

插入排序

- 插入排序: 选择一个基准元素, 将该元素与前面的元素比较, 插入到合适的位置
- 时间复杂度: O(n^2), 空间复杂度: O(1), 稳定性: 稳定

快速排序

- 快速排序: 选择一个基准元素, 将小于基准的元素放到左边, 大于基准的元素放到右边, 递归处理左右两部分
- 时间复杂度: O(nlogn), 空间复杂度: O(logn), 稳定性: 不稳定

归并排序

- 归并排序: 将数组分为两部分, 每部分继续分为两部分, 直到每部分只有一个元素, 再将两部分合并
- 时间复杂度: O(nlogn), 空间复杂度: O(n), 稳定性: 稳定

堆排序

- 堆排序: 将数组构建成最大堆, 每次取出堆顶元素, 重新调整堆

- 时间复杂度: O(nlogn), 空间复杂度: O(1), 稳定性: 不稳定

桶排序

- 桶排序: 将数组分为若干个桶, 每个桶内部排序, 最后合并桶
- 时间复杂度: O(n+k), 空间复杂度: O(n+k), 稳定性: 稳定

计数排序

- 计数排序: 统计每个元素出现的次数, 再根据次数排序
- 时间复杂度: O(n+k), 空间复杂度: O(n+k), 稳定性: 稳定

基数排序

- 基数排序: 从低位到高位依次排序, 每次按照某一位排序
- 时间复杂度: O(n*k), 空间复杂度: O(n+k), 稳定性: 稳定

3. 分治算法

- 分治算法: 将问题分解为若干个子问题, 分别求解子问题, 再合并子问题的解
- 时间复杂度: O(nlogn), 空间复杂度: O(logn)

4. 动态规划

- 动态规划: 将问题分解为若干个子问题, 求解子问题, 再根据子问题的解求解原问题
- 时间复杂度: O(n^2), 空间复杂度: O(n^2)

5. 贪心算法

- 贪心算法: 每次选择最优的解, 直到得到最终解
- 时间复杂度: O(n), 空间复杂度: O(1)
- 用途: 解决最优化问题, 例如最小生成树, 最短路径等