一些算法的时间复杂度

2024年12月30日 星期一　　19:05

3. 各排序算法的比较

排序算法	时间复杂度（最坏/平均/最好）	空间复杂度	稳定性	特点与适用场景
冒泡排序	O(n^2) / O(n^2) / O(n)	O(1)	稳定	简单实现，适合小规模数据或数据接近有序的场景
选择排序	O(n^2)	O(1)	不稳定	简单但效率低，不适合大数据
插入排序	O(n^2) / O(n^2) / O(n)	O(1)	稳定	适合少量数据或几乎有序的数组
归并排序	O(n log n)	O(n)	稳定	适合大数据、需要稳定性的场景
快速排序	O(n^2) / O(n log n) / O(n log n)	O(log n)	不稳定	高效通用的排序算法，适合大规模数据
堆排序	O(n log n)	O(1)	不稳定	用于堆结构的场景，内存使用较少
计数排序	O(n + k)	O(n + k)	稳定	适用于数据范围较小且整数数据的场景
桶排序	O(n + k)	O(n + k)	稳定	适合数据均匀分布的场景
基数排序	O(n * k)	O(n + k)	稳定	适合多位数值或字符串的排序

	查找	更新	插入	删除
数组	O(1) 知道index的情况下	O(1)	O(n)	O(n)
链表	O(n)	O(1)	O(1)	O(1)

最坏/平均情况

	查找	插入	删除
有序数组	Θ(logn)/Θ(logn) 查找为二分查找	Θ(n) / Θ(n)	Θ(n)/Θ(n)
有序链表	Θ(n)/Θ(n)	Θ(n)/Θ(n)	Θ(n)/Θ(n)
跳表	Θ(n)/Θ(logn)	Θ(n)/Θ(logn)	Θ(n)/Θ(logn)
哈希表	Θ(n)/Θ(1)	Θ(n)/Θ(1)	Θ(n)/Θ(1)

二叉树：遍历：时间和空间都为O(n)（四种遍历方式都一样）

访问一个节点：Θ(1)

大/小根堆：初始化：O(nlogn) 算法复杂性上限　Θ(n) 实际的复杂性

插入：O(logn)

删除：O(logn)

左高树：合并：O(log(mn))

初始化：O(n)

构造霍夫曼树：O(nlogn)

二叉搜索树：查找：O(h)　h是树的高度

插入&删除：O(h)　O(logn)　n是节点数

最坏/平均情况

	搜索	插入	删除
二叉搜索树	n / logn	n/logn	n/logn
AVL树	logn/logn	logn/logn	logn/logn
B-树	logn/logn	logn/logn	logn/logn

拓扑排序：邻接矩阵：O(n²)　n表示顶点数，e表示边数

邻接链表：O(n+e)

Dijkstra 算法：O(n²logn)

Kruskal 算法：O(nteloge)

Prim 算法：O(n²)

Floyd 算法：递归：O(3ⁿ)

迭代：Θ(n³)

	邻接矩阵	邻接链表
BFS	O(sn)	$O(\sum_i d_i^{out})$　s为被标记顶点数
DFS	O(sn)	$O(\sum_i d_i^{out})$

计算入度

表示方法	时间复杂度	说明
邻接表	O(E) 或 O(V+E)	O(E)指只计算一个顶点的入度；O(V+E)考虑了遍历所有顶点寻找入边的情况，最坏情况相当于遍历所有边。预先计算入度则为O(1)
邻接矩阵	O(V)	遍历矩阵的一列