易错遗忘点

2021年11月2日 21:13

关于排序的性能比较:

从时间复杂度来看:
插入排序，简单选择排序，冒泡排序的平均时间复杂度为$O(n^2)$,
快速排序的平均时间复杂度为$O(n\log_2 n)$,
堆排序的平均时间复杂度也为$O(n\log_2 n)$,
二路归并排序的平均时间复杂度为$O(n\log_2 n)$。

是否受初始状态影响:
插入排序受初始状态影响，在最好的情况下，简单选择排序的时间复杂度可以达到O(n)（初始序列已经有序，只需要每次插入最后一个）。
冒泡排序的最好时间复杂度也可以达到O(n)（初始序列已经有序，扫描一遍之后发现不需要交换任何两个元素之间的位置）。
折半插入排序只是减少了比较的次数，没有减少移动的次数，因此时间复杂度仍为$O\left(n^2\right)$。
快速排序受初始状态影响，最坏的情况（初始序列已经有序，同时每次选择第一个元素作为枢轴元素）时最坏时间复杂度为$O\left(n^2\right)$。
堆排序算法不受初始状态影响，最好、最坏、平均的时间复杂度均为$O(n\log_2 n)$。
二路归并算法与初始序列无关，最好、最坏、平均的时间复杂度均为$O(n\log_2 n)$。
简单选择排序不受初始状态的影响，任何情况下的时间复杂度都是$O\left(n^2\right)$。

从空间复杂度来看:
插入排序，冒泡排序，简单选择排序，希尔排序，推排序的空间复杂度均为O(1);
快速排序的空间复杂度受枢轴元素的影响，最好的情况下为$O(\log_2 n)$,最坏的情况下为O(n);
二路归并算法需要一个数组空间用于元素复制，故空间复杂度为O(n)。

是否稳定:
插入排序，冒泡排序，归并排序，基数排序是稳定的。
希尔排序，快速排序，简单选择排序，堆排序均是不稳定的。

关于图的各种应用算法的复杂度总结:

1.求最小生成树（带权连通无向图）:

算法	时间复杂度	适合稠密/稀疏
prim算法	$O(\|v^2\|)$	与顶点有关，稠密
Kruskal算法	$\|E\|\log\|E\|$	与边有关，稀疏

2.求最短路径（带权有向图）

	时间复杂度	适用于负权值
Dijkstra算法	$O(\|v^2\|)$	否
Floyd算法	$O(\|v^3\|)$	是

3.拓扑排序（AOV，有向无环图）

算法	时间复杂度
拓扑排序算法	$O(\|v\|+\|E\|)$

关于哈希表的解决冲突方法

1. 开放定址法

$H_i=(H(key)+d_i)\ \ MOD\ \ m$

H（key）为哈希函数，m为散列表表长，di为增量序列。

①. 线性探测再散列法

$d_i = 0, 1, 2, \cdots, m-1$

②. 二次探测再散列法

$d_i = 0^2, 1^2, -1^2, 2^2, -2^2, \cdots, k^2, -k^2$

m必须是一个可以表示成4m+3的素数

③. 再散列法

$d_i = \mathrm{Hash}_2(\mathrm{key})$

④. 伪随机序列法

2. 拉链法

将所有关键字同义词的记录存储在同一单链表中，并且按关键字有序排列。

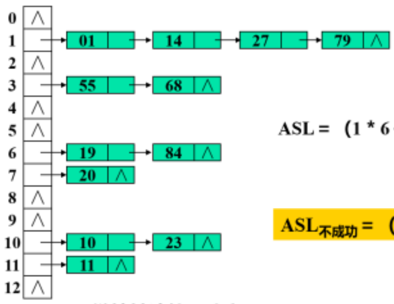
哈希表的查找性能

平均查找长度ASL取决于哈希函数、处理冲突的方法以及装填因子

$$装填因子\alpha = \frac{表中填入的记录数}{哈希表的表长}$$

装填因子越大，哈希表发生冲突的可能性越大，平均查找长度也越大。

例1：关键字序列（19 14 23 01 68 20 84 27 55 11 10 79），哈希表长为16，哈希函数H（key）＝key MOD 13。



ASL＝（1 * 6 + 2 * 4 + 3 + 4）/12=1.75

$ASL_{不成功}$＝（1+5+1+3+1+1+3+2+1+1+3+2+1）/13=1.92

链地址法处理冲突

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	14	01	68	27	55	19	20	84	79	23	11	10			

线性探测再散列法处理冲突

（1）ASL＝（1 * 6 + 2+3 * 3 + 4 + 9）/12

$ASL_{不成功}$＝（1 + 13+12+11+10+9+8+7+6+5+4+3+2）/13 = 7.1

关于广度优先遍历和深度优先遍历的算法复杂度

方法	邻接矩阵	邻接表
广度优先	时间：O($\|v\|^2$) 空间：O($\|v\|$)	时间：O($\|v\| + \|E\|$) 空间O($\|v\|$)
深度优先	时间：O(($\|v\|^2$) 空间：O($\|v\|$)	时间：O($\|v\| + \|E\|$) 空间：O($\|v\|$)

树的应用

1. 二叉排序树（BST）

1) 二叉排序树的定义

①. 若左子树非空，则左子树上所有结点的值都小于根节点

②. 若右子树非空，则右子树上所有结点的值都大于根节点
③. 左、右子树也分别是一棵二叉排序树

2) 二叉排序树的查找
3) 二叉排序树的插入
4) 二叉排序树的构造
5) 二叉排序树的删除
   ①. 若删除的是叶节点，则直接删除
   ②. 若删除的不是叶节点，则用它的直接前驱或者直接后继替代
6) 二叉排序树的查找效率
   最好的情况下可以达到$O\left(\log_2 n\right)$,最坏的情况下平均查找长度为O(n)
7) 二叉排序树与二分查找的判定树的区别：
   ①. 二分查找的判定树唯一，而二叉排序树的查找不唯一
   ②. 二叉排序树的插入，只需要修改指针，平均时间复杂度为$O(log_2 n)$；
      二分查找的对象是有序顺序表时，插入和删除的代价都是O(n)
   若有序表是静态查找表，则采用顺序表作为存储结构，采用二分查找进行查找；
   若有序表是动态查找表，则采用二叉排序树作为逻辑结构

2. 平衡二叉树（AVL）

1) 平衡二叉树的定义
2) 平衡二叉树的插入
   ①. LL
   ②. RR
   ③. LR（左孩子的右子树上插入了新的结点）
   ④. RL（右孩子的左子树上插入了新的结点）
3) 平衡二叉树的查找
   ①. 深度为h的平衡二叉树中含有的最少结点数——nh = nh-1+nh-2+1(n0=0,n1=1,n2=2···)
   ②. 含有n个结点的平衡二叉树的最大深度为（$log_2 n$），平衡二叉树的平均查找长度为$(log_2 n)$

3. 哈夫曼树

1) 哈夫曼树的定义
   带权路径长度
      结点的：从树的根到任意结点的路径长度与该节点权值的乘积，称为该结点的带权路径长度
      树的：所有叶节点的带权路径长度称为树的带权路径长度
2) 哈夫曼树的构造

   易错遗忘点

   ①. 将这n个结点分别作为n棵只含一个结点的二叉树，构成森林F
   ②. 构造一个新节点，从F中选取两棵根节点权值最小的树作为新结点的左、右子树，并且将新节点的权值置为左、右子树上根节点的权值之和
   ③. 从F中删去刚才选中的两棵树，同时将新得到的树加入F
   ④. 重复步骤2）和3）直到F中只剩下一棵树为止
★3) 哈夫曼树的特点

   易错遗忘点

   ①. 每个初始结点最终都会称为叶节点
   ②. 构造过程中一共创建了n-1个新节点，哈夫曼树的总结点个数为2n-1
   ③. 哈夫曼树中不存在度为1的结点
4) 哈夫曼编码

有关图的一些概念

1. 完全图
    1) 无向图：含有n(n-1)/2条边的无向图
    2) 有向图：含有n(n-1)条弧的有向图
2. 生成子图
   包含所有顶点的子图称为生成子图
3. 连通、连通图、连通分量
    1) 连通：两个顶点由路径存在
    2) 连通图：任意两个顶点之间连通
    3) 连通分量：极大连通子图
4. 生成树
   包含全部顶点的极小连通子图
5. 顶点的度、入度、出度
    1) 无向图：所有顶点的度之和等于边数的两倍
    2) 有向图：全部顶点的入度和出度之和相等
6. 路径、路径长度和回路
    1) 路径：两个顶点之间的一系列顶点序列，也包括相关联的边
    2) 路径长度：路径中边的数目
    3) 回路：第一个顶点和最后一个顶点相同的路径
7. 简单路径、简单回路
    1) 简单路径：顶点不重复出现的路径
    2) 简单回路：除了第一个和最后一个顶点之外，其余顶点不重复出现的回路

图的应用

1. 最小生成树
    1) 生成树的定义
       对于一个生成树，砍去它的一条边，就会变成一个非连通图；增加一条边，就会形成回路
    2) 最小生成树的定义
       边的权值最小的生成树
    3) 最小生成树的性质
        ①. 最小生成树不唯一；除非图中各边的权值互不相等
        ②. 最小生成树的边的权值之和是唯一的
        ③. 最小生成树的边数等于顶点数减一
    4) 算法
        ①. Prim算法
            i. 初始时任选一个顶点加入树T，
            ii. 选择一个与当前T中顶点集合距离最近的顶点，并将该顶点和相应的边加入T
            iii. 重复2)，最终一定可以得到一棵边为n-1的最小生成树
        ②. Kruskal算法
            i. 初始时只有n个顶点而无边的非连通图，每个顶点自成一个连通分量
            ii. 按照边的权值由小到大的顺序，不断选取当前未被选取过且权值最小的边
            iii. 若该边加入T后不构成回路，则将该边加入T，否则舍弃此并寻找下一条权值最小的边
            iv. 重复2）和3）直到T中所有结点都在一个连通分量上
2. 最短路径
   性质：两点之间的最短路径也包含了路径上其他顶点的最短路径
    1) Disjkstra算法

顶点	第一轮	第二轮	…第k轮
2			
3			

4			
5			
集合	{1，…}		

2) 注意事项

   对负权值的边，Dijkstra算法不适用

3) Floyd算法

   逐步迭代方阵，依次加入顶点v_i（i = 0, 1, 2…n-1）

   适用于带负权值的边，不允许有包含带负权值的边组成的回路

3. 拓扑序列
   1) 从AOV网中选取一个入度为0的顶点并输出
   2) 从网中删除该顶点和所有以它为起点的有向边
   3) 重复上述步骤直到AOV为空，或者当前网中不存在入度为0的结点（图中必然有环）
4. 关键路径
   1) 求解事件的最早发生时间和最晚发生时间

时间	v1	v2	v3…
ve(i)			
vl(i)			

   2) 求解活动的最早发生时间和最晚发生时间
      ①. 活动的最早开始时间：该活动弧的起点所表示的事件的最早开始时间
      ②. 活动的最晚开始时间：该活动弧的终点所表示的事件的最晚开始时间—活动时间
   3) 注意事项
   i. 关键路径上的所有活动都是关键活动，可以通过缩短关键活动的时间来缩短整个工程的工期，但是不能无限缩短
   ii. 关键路径不唯一，必须缩短所有关键路径上的共同路径才能缩短工程的工期

栈和队列

1. 栈的数学性质

   n个不同元素进栈，出栈元素不同排列的个数为$\frac{1}{n+1}C_{2n}^{n}$。

2. 循环队列的判空、判满

   初始时：Q.front = Q.rear = 0

   入队时：Q.front = (Q.front+1)%Q.front

   出队时：Q.rear = (Q.rear-1)%Q.rear

   队列长度：Length = (Q.rear - Q.front + MaxSize)%Maxsize

   队空：Q.front = Q.rear

   判断队满：

   ①. 牺牲一个单元

   (Q.rear+1) % Maxsize = Q.front

   ②. 增加一个表示元素个数的数据成员

   Q.length = Maxsize

   ③. 增加一个tag数据成员

3. 栈和队列的应用

   ①. 栈在括号匹配中的应用
      1) 初始设置一个空栈，顺序读入括号
      2) 若是右括号，则要不然和栈顶的元素匹配，将栈顶元素弹出，要不然就是和栈顶元素不匹配，出现了不合法的情况。
      3) 若是左括号，则压入栈称为新的栈顶元素，算法结束时，如果括号匹配则栈为空，或者是括号序列不匹配

   ②. 栈在求值表达式中的应用
      1) 中缀表达式转后缀表达式
         i. 建立一个运算符栈
         ii. 从左到右顺序读取表达式
            a) 如果读取到的是数字，则直接输出到后缀表达式
            b) 如果读取到的是"("，则直接压入栈

c) 如果读取到的是"0"，则依次弹出栈顶元素知道弹出"("为止
d) 如果读取到的是除了"("以外的其他运算符，则将其与栈顶元素的优先级进行比较，如果大于栈顶元素的优先级，则直接压入栈，否则弹出栈顶元素，直到栈为空或者栈顶元素优先级低于该运算符。
iii. 如果表达式已经读取完，但是栈中还有元素，则将栈中所有元素依次弹出直到栈为空。

2) 中缀表达式转前缀表达式
i. 建立一个运算符栈
ii. 从右向左读取表达式
a) 如果读取到的是数字，则直接输出到前缀表达式
b) 如果读取到的是")"，则直接入栈
c) 如果读取到的是"("，则将栈顶运算符依次弹出并输出到前缀表达式，直到弹出"("为止。
d) 如果读取到的是非括号运算符，则将其与栈顶运算符的优先级进行比较，如果该运算符的优先级不小于栈顶运算符的优先级，则直接入栈；否则弹出栈顶运算符并输出到前缀表达式，继续比较新的栈顶元素和该运算符的优先级，直到栈为空或者新的栈顶运算符的优先级低于当前读取的运算符的优先级。
iii. 如果表达式读取完但是栈不空，则将栈中的所有元素依次弹出直到栈为空。

3) 通过后缀表达式计算表达式的值
i. 若该项是操作数，则压入栈中，
ii. 若该项是操作符，则从栈中连续退出两个操作数Y和X并进行运算，并将运算结果重新压入栈（注意是第二个栈顶元素<OP>第一个栈顶元素）
iii. 当所有的表达式都处理完之后，栈中的值就是最后表达式的值

③. 栈在递归当中的作用
1) 递归表达式
2) 递归出口

④. 队列在层次遍历中的应用
1) 如果队列不为空，首先将根节点入队
2) 然后依次输出队首元素，如果该元素有左孩子，则左孩子入队；该元素有右孩子，则右孩子入队
3) 重复第二个步骤直到队列为空。

⑤. 队列在计算机系统中的应用
1) 解决主机与外部设备之间的速度不匹配
2) 解决由多用户引起的资源竞争问题

树的一些术语和性质

1. 度
①. 一个结点的孩子个数就是这个结点的度
②. 树中结点的最大度数称为树的度
③. 树的结点个数与树的结点的度之和的关系
结点的个数 = 所有结点的度之和+1

2. 深度、高度、层次
①. 深度：从根节点向下逐层累加
②. 高度：从叶节点向上逐层累加
③. 层次：根节点为第一层，依次向下增加

3. 路径和路径长度
①. 路径：两个结点之间所经过的节点序列
②. 路径上所经过的边的个数

4. 树的性质
①. 一般树的性质
树的结点数=树的所有结点的度数之和+1
②. 特殊二叉树的性质
1) 满二叉树

i. 高度为h的满二叉树的结点数为2^h-1
ii. 第k层的结点数为2^(k-1)
iii. 所有叶子结点都集中在最下面一层
iv. 除叶子节点意外每个结点的度都为2
v. 编号为i的结点的双亲为$\lfloor \frac{i}{2} \rfloor$，如果有孩子，则左孩子为2i，则右孩子为2i+1

2) 完全二叉树

i. i<$\lfloor \frac{n}{2} \rfloor$，则i为分支结点，否则i为叶节点
ii. 叶子节点只可能在最下面两层出现
iii. 度为1的结点最多只有一个
iv. n为奇数，则每个结点都有左孩子和右孩子，否则n/2的结点只有左孩子，没有右孩子

3) 二叉树的性质

i. n0 = n2+1
ii. 具有n个结点的完全二叉树的高度为$\lceil \log_2(n+1) \rceil$，或者是$\lfloor \log_2 n \rfloor + 1$

③. 二叉树的链式存储结构

一共有n+1个空链域

5. 森林转化成二叉树

①. 将森林中的每棵树都转换成二叉树
②. 将每棵的根看作兄弟关系，每两棵树根中间加一连线
③. 以第一棵树的根为轴心，顺时针旋转45°

各种算法表述

1. 邻接矩阵存储

易错遗忘点

2. 邻接表法

易错遗忘点

3. 广度优先遍历的思想

易错遗忘点

4. 深度优先遍历的思想

易错遗忘点

5. prime算法

易错遗忘点

6. 克鲁斯卡尔算法

易错遗忘点

易错遗忘点

7. 迪杰斯特拉算法

易错遗忘点

8. 拓扑排序算法