Data Structure-Note

made by LaTeX

NP_123

2022 年 5 月 14 日

np123greatest@gmail.com

Linear Structure

顺序结构线性表

- 插入，删除

> 1. 插入前检查是否满
> 2. 判断下标合理性
> 3. 在第 i 个位序上插入 X，从 a_i 到 a_n 都要向后移动一个位置，**一共移动** $n-i+1$ **个元素**平均移动次数为 $\frac{n}{2}$ 时间复杂度 $O(n)$

> 1. 删除前检查是否空
> 2. 判断下标合理性
> 3. 在第 i 个位序上删除 X，从 a_{i+1} 到 a_n 都要向前移动一个位置，**一共移动** $n-i$ **个元素**平均移动次数为 $\frac{n-1}{2}$ 时间复杂度 $O(n)$

链式结构线性表

1. 为了避免插入头节点时改变表头，可以为链表增加一个空的“头节点”
2. 在单链表上插入、删除一个节点，必须知道其前驱节点

广义表与多重链表

堆栈

- 概念
  顺序存储：Top 指向-1，表示空栈
  链式存储：插入和删除操作只能在链栈的栈顶进行
- 中缀转后缀表达式

> 1. 如果遇到空格符认为是分隔符，不处理
> 2. 若遇到运算数，直接输出
> 3. 若遇到左括号，则压入栈中
> 4. 若遇到右括号，将栈顶的运算符弹出并输出，直到左括号（左括号弹出，但不输出）
> 5. 若遇到运算符：
>   (a) 若运算符优先级大于栈顶运算符，压栈

(b) 若优先级小于等于栈顶运算符，将栈顶的运算符弹出并输出，继续比较, 最后自己入栈

6. 处理完毕，则将栈顶的运算符一并输出

队列

若用队列元素个数 size 代替一般循环队列中的 front 和 rear 指针，则能容纳的元素数量从 m-1 提升为 m

1. 队满：$(Rear+1)\%Size = Front$
2. 队空：$Rear = Front$
3. 插入新元素的位置 $(Rear+1)\%Size$

$\star$ 双端队列

- Push
- Pop
- Inject
- Eject

Tree

离散数学基础知识

- 握手定理：度数之和 = 边数的两倍
- n 阶，m 条边的树：m=n-1
- BT 的深度小于等于节点数 N，平均深度是 O($\sqrt{N}$)
- n 叉树的时候：(B：边数 n：节点数)

$$n = \sum_{i=0}^{m} n_i \tag{1}$$

$$B = n - 1 \tag{2}$$

$$B = \sum_{i=1}^{m} i \times n_i \tag{3}$$

联合 (1)(2)(3) 整理得$n = 1 + \sum_{i=1}^{m} i \times n_i = \sum_{i=0}^{m} n_i = n_0 + \sum_{i=1}^{m} n_i$

即

$$n_0 = \sum_{i=1}^{m} (i-1) \times n_i + 1 \tag{4}$$

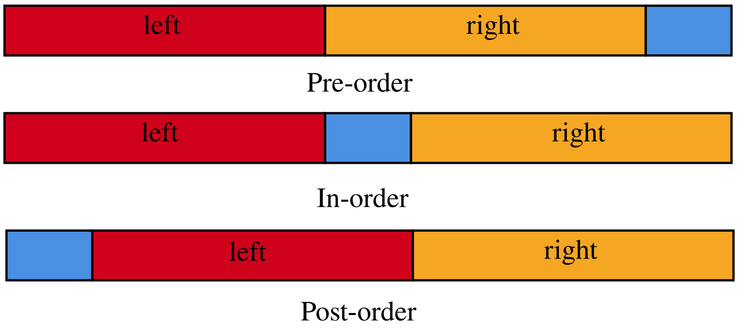
Binary Tree

第 i 层最大节点数 2^{i-1} $(i >= 1)$
深度为 k 的二叉树最大节点总数 $2^k - 1$ $(k >= 1)$
叶节点个数 = 度为 2 的非叶节点个数 +1
具有 n 个节点的完全二叉树的深度 k 为 $\lfloor \log n \rfloor + 1$

- 存储结构：顺序存储（起始下标为 1），链表存储
- 前序，中序，后序遍历

递归算法
$\star$ 非递归算法：遇到节点压入栈，再遍历左子树。遍历节点

- 层序遍历-队列
- 求二叉树高度
- 由两种遍历确定二叉树



Binary Search Tree

- 定义

> 左子树键值**小于**根节点的键值
> 右子树键值**大于**根节点的键值
> 左、右子树都是二叉搜索树

- 动态查找
- 搜索树的插入，删除从右子树找最小的元素，从左子树找最大元素（被选择的节点必定最多只能有一个孩子）

AVL

- 定义

> BF(T)= $h_L - h_R \ = \{-1, 0, 1\}$
> 旋转：L,R,LR,RL

Heap

> - 插入
> - 删除从根节点开始，用最大堆中的最后一个元素向上过滤下层节点
> - 创建从最后一个结点的父节点开始，下滤到根节点 1
>
> 时间复杂度 O(NlogN)

Huffman Tree

- 带权路径长度 (根节点权值为 0)

$$WPL = \sum_{k=1}^{n} W_k l_k$$

- 算法复杂度

> 调整最小堆 O(N)
> 2(N-1)+1 个删除：O(NlogN)
> N-1 个插入：O(NlogN)

Set

- 查找元素，并运算
- 按秩合并
  小集合并入大集合
- 路径压缩
  从 X 到根节点上所有的节点都变成了根节点的孩子

Hash Search

基本概念

装填因子 $\alpha = n/m$：填入的元素为 n，散列表空间大小为 m

一个好的散列函数考虑下列两个因素：

1. 计算简单，以便提高转换速度
2. 关键词对应的地址空间分部均匀，以尽量减少冲突

影响产生冲突的多少有以下三个因素

1. 散列函数是否均匀
2. 处理冲突的方法
3. 装填因子 α

平均查找程度 ASL：查找所有节点的次数的平均值

构造方法

- 数字关键词
  1. 直接定址法　$h(key) = a \times key + b$　(a、b 为常数)
  2. 除留余数法　$h(key)$ =key mod p
  3. 数字分析法
- 字符串关键词
  1. ASCII 码加和法　$h(key) = (\sum key[i])$ mod TableSize
  2. 前 3 个字符移位法　$h(key)$ = (key[0]+key[1] ×27+key[2]×27^2) mod TableSize
  3. 移位法　$h(key) = \sum_{\mathrm{i=0}}^{\mathrm{n-1}} \mathrm{key[n-i-1\times 32^i]}$ mod TableSize

处理冲突的方法

- 开放定址法　$h_i(key) = (h(key) + d_i)$ mod TableSize　(1≤ i <TableSize)
  1. 线性探测法　$d_i = i$
     一次聚集：通过线性探测法造成的聚集
  2. 平方探测法　$d_i = \pm i^2$
     二次聚集：通过平方探测法造成的聚集
     “懒惰删除”：需要添加一个“删除标记”，而不是真正的删除

3. 双散列探测法　　$d_i = i \times h_2(key)$
   一般形如 $h_2(key) = p-$(key mod p) 这样的函数会有良好效果，p 是小于 TableSize 的素数。
   选用一个素数作为 TableSize 也同样重要，否则可能探测不到所有储存单元
4. 再散列法
   再散列：当装填因子 α 过大时，解决方法是加倍扩大散列表
   再散列需要新建一个两倍大的散列表，并将原表的数据重新计算分配到新表去

- 分离链接法 ez

Graph

离散数学基础知识

- 无向完全图：一共 $n(n-1)/2$ 条边　　　有向完全图：一共 $n(n-1)$ 条边
- 最大度 $\Delta(D)$，最小度 $\delta(D)$
- 连通图/强连通图：无向图/有向图从一个顶点到另一个顶点是相通的
  连通分量/强连通分量：极大连通子图

Adjacency Matrix

- 优势
  容易确定途中任意两个顶点是否有边相连
- 劣势
  确定一共有多少边，需要对每个元素检测，时间复杂度 $\Theta(|V|^2)$
  花费 $\Theta(Nv^2)$ 即 $\Theta(|V|^2)$ 得储存空间（对于稀疏图浪费空间）

Adjacency Lists

需要有 $|V|$ 个头节点和 $2|E|$ 个表边节点
有向图：为了求入度方便，建立逆邻接表：对每个顶点 v_i 建立一个链接以 v_i 为头的弧的链表

$$\text{建立邻接表时} \begin{cases} \text{输入的顶点信息为编号} & O(|V|+|E|) \\ \text{否则，需要查找所在位置} & O(|V|\cdot|E|) \end{cases}$$

DFS，BFS

邻接矩阵：查找所顶点的邻接点 $O(|V|^2)$
邻接表：查找临界点 $O(|E|)$，遍历图 $O(|V|+|E|)$

Minimum Spanning Tree/Path

- Prim:$O(|V|^2)$　　　适合稠密图
- Kruskal:$O(|E|\cdot\log(|V|))$　　　适合稀疏图
- Dijkstra: 稠密图:$O(|V|^2)$，稀疏图可改进为 $O(|E|\cdot\log(|V|))$
- Floyd:$O(|V|^3)$

Topological Sorting

邻接矩阵：$O(|v|^2)$，邻接表：$O(|E|+|V|)$