

# 5 二叉树

## 5.1 基本概念

### Definitions

- 节点状态：空(树叶), 右空, 左空, 都不空(满)
- 节点集合：内部节点/分支节点(非树叶), 外部节点(树叶)
- **满二叉树**：树上只有叶子和满节点
- **完全二叉树**：除了倒数两层都满, 最后一层叶子从左向右填充
  - 只有倒数两层有树叶.
  - **路径长度**和**最短**.
- **扩充二叉树**：出现空指针时, 增加一个树叶.
  - 扩充二叉树是**满二叉树**.
  - 新增空树叶的个数等于原来二叉树结点个数加1
  - 外部路径长度 = 内部路径长度 + 2 内部节点个数 (归纳法证明)

### Properties

- 高度 = 深度 + 1
- 叶子数 = 满节点数 + 1
  - 非空满二叉树情形下的该性质称为**满二叉树定理**, 此时叶子数 = 内部节点数 + 1
  - **满二叉树定理推论**：空子树数 = 节点数 + 1
- 完全二叉树高度 =  $\lceil \log_2(\text{节点数}+1) \rceil$ , 节点数  $\leq 2^{\text{二叉树高度}} - 1$

## 5.2 二叉树的遍历

### DFS遍历

```
f(x,y) :  
    if (y=前序) 访问x  
    f(x->leftchild,y)  
    if (y=中序) 访问x  
    f(x->rightchild,y)  
    if (y=后序) 访问x
```

- 时间复杂度  $O(n)$ , 空间复杂度  $O(\log n)$
- 前序+后序不能唯一确定树, 中序+前序或后序可以

## 非递归DFS

- 用栈实现
- 前序

```

栈S, x=root
while(x):
    访问x
    if(x->rightchild != NULL) S.push(x->rightchild)
    if(x->leftchild != NULL) x=x->leftchild
    else x=S.pop()

```

- 中序

```

栈S, x=root
while(x || !S.empty()):
    if(x)
        S.push(x)
        x=x->leftchild
    else
        x=S.pop()
        访问x
        x=x->rightchild

```

- 后序

```

栈S, x=root
while(x || !S.empty()):
    while(x):
        x.tag=left
        S.push(x)
        x=x->leftchild
    x=S.pop()
    if(x.tag=Left)
        x.tag=right
        S.push(x)
        x=x->rightchild
    else
        访问x
        x=NULL

```

- 时间复杂度  $O(n)$ , 空间复杂度最好  $O(\log n)$  最坏  $O(n)$  (取决于树的高度)

## BFS遍历

- 从根节点开始, 自上而下逐层遍历, 用队列实现
- 时间复杂度  $O(n)$ , 空间复杂度最好  $O(1)$  最坏  $O(n)$  (取决于树的最大宽度)

## 5.3 二叉树的存储结构

- 二叉链表: 本身数据和left和right指针.
- 三叉链表: 增加一个指针parent.
- 可以递归或非递归地寻找父节点or兄弟节点

### 空间开销分析

$$\alpha(\text{存储密度}) = \frac{\text{数据本身存储量}}{\text{结构占用存储量}}, \quad \gamma(\text{结构性开销}) = 1 - \alpha$$

e.g. 二叉链表每个结点存两个指针, 一个数据域, 结构性开销  $\frac{2p}{2p+d}$

如何节省?

- union联合类型
- 子类分别实现内部节点与叶结点, 用虚函数区分
- 利用节点指针的一个空闲位标记类型

- 利用指向叶的指针或者叶中的指针域来存储该叶结点的值

## 完全二叉树的下标公式

假定根为 0, 对于节点  $i$

左孩子	右孩子	父节点	左兄弟	右兄弟
$2i + 1$	$2i + 2$	$\lfloor (i - 1) / 2 \rfloor$	$i - 1$	$i + 1$

## 5.4 二叉搜索树 (BST)

也称二叉排序树

### Definition

- 或者是空树.
- 对于任何值为  $k$  的结点, 该结点的左子树中结点值都小于  $k$ ;
- 该结点右子树的结点值都大于  $k$ ;
- 左右子树也为二叉搜索树.
- 按照**中序周游**将各结点打印出来, 将得到**由小到大的**排列.

### Methods

- 检索  $x$ : 与结点比较, 区分检索左子树还是右子树, 递归.
  - 时间复杂度  $O(\log n)$
- 插入  $x$ : 先检索, 如果找到了不允许插入, 否则一定找到一个空子树, 在该位置插入一个新叶
- 删除  $x$ : 左子树替代  $x$ , 右子树移植在左子树最大值节点上 or 删除左子树最大值节点, 最大值替代  $x$ 
  - 后者可以防止高度失衡

## 5.5 堆与优先队列

### 最小堆

序列  $K_0, K_1, \dots, K_{n-1}$ , 有如下特性:

$$K_i \leq \min\{K_{2i+1}, K_{2i+2}\}$$

- 由完全二叉树表示, 即为父节点不大于子节点
- 局部有序, 堆不唯一

## 建堆

自底向上逐步把以子树调整成堆, 注意保子树性质

```
Heapify(array):
    n = array.length()
    for (i = n/2-1; i>=0; i--): \\遍历内部节点
        Siftdown(array, i, n)

Siftdown(array, index, n):
    smallest = index
    left = 2*index+1
    right = 2*index+2
    if (left<n AND array[left]<array[smallest])
        smallest = left
    if (right<n AND array[right]<array[smallest])
        smallest = right
    if smallest != index:
        SWAP(array[smallest], array[index])
        Siftdown(array, smallest, n) \\保子树的最小堆性质
```

## 插入

新元素被加入到堆的末尾, 然后更新树以恢复堆的次序

```
Insect(array, value):
    array.append(value)
    Siftup(array, array.size()-1)

Siftup(array, index):
    while (index > 0):
        parent = (i-1)-2
        if array[index] < array[parent] :
            swap(array[index], array[parent])
            index = parent
        else
            break
```

## 删除(最小值)

```

DeleteMin(array):
    SWAP(array[0], array[array.size()-1])
    min_value = array.pop()
    SiftDown(array, 0, array.size())
    return min_value

```

## 效率分析

$$\sum_{i=1}^{\log n} (i-1) \frac{n}{2^i} = O(n)$$

- 插入,删除最小值,删除普通节点的平均和最差时间代价都是  $O(\log n)$

堆可以用于实现优先队列

## 5.6 Huffman编码树

- **等长编码**
  - 表示  $n$  个不同的字符需要  $\log_k n$  位.
  - 字符的使用频率相等.
- 频率不等的字符, 可以利用字符的出现频率来编码.
  - 经常出现的字符的编码较短, 不常出现的字符编码较长.
- **前缀编码**: 任何一个字符的编码都不是另外一个字符编码的前缀
  - 保证编码映射是双射

## 定义

- **哈夫曼树 (或称最优二叉树)**: 具有最小带权外部路径长度的二叉树

$$\min \sum_{i \in [n]} w_i \cdot l_i$$

权重即字符出现频数, 外部路径保证是前缀编码

### 建树

按权从小到大排列字符, 每次取前两个权标记为分支节点的两个孩子, 将新节点赋权后放回序列, 重复步骤

### 译码

从左到右逐步判别代码串, 直至确定一个字符

如何证明建树过程实现了"最小外部路径权重"?

答: 归纳证明, 考虑最小权的两个叶子(他们是兄弟节点)

## 效率分析

- Huffman编码压缩比率随字符频率分布变化
- 外部数目不能构成满 $k$ 叉Huffman树时,需要添加权重为0的虚节点
  - 每个内部节点提供了  $b - 1$  个"外-内"度数