第六课-树、二叉树、二叉搜索树，堆，图

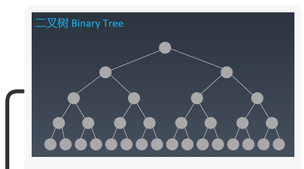
别人家的康少

tree

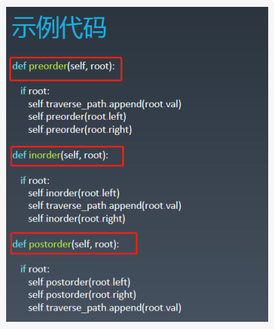
- Linked List Linked List 是特殊化的 Tree
- Tree 是特殊化的 Graph Graph
- 示例代码

图 Graph

二叉树



- 二叉树遍历
  1. 前序（ Pre -order ）： 根 -左-右
  2. 中序（ In -order ）： 左 -根-右
  3. 后序（ Post Post -order ）： 左 -右-根
- 示例代码



二叉搜索树

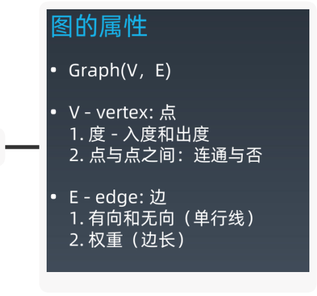
二叉搜索树， 也称二叉 排序 树、有序二叉树（ Ordered Binary Tree Binary TreeBinary Tree ）、 排 序二叉树（ Sorted Binary Tree Binary Tree Binary Tree Binary Tree），是指一棵空树或者具有下列性质的二叉 树：

1. 左子树上 所有结点 的值 均小于它根结点；
2. 右子树上 所有结点 的值 均大于它根结点；
3. 以此类推：左、右子树 也分别为二叉查找。 （这就是 重复性！ ）

- 中序遍历：升排列
- 二叉搜索树常见操作
  1. 查询
  2. 插入新结点（创建 ）
  3. 删除 — 如果删掉以后要选择一个替换 — 右边 — 最靠近的一个节点替换 — Logn
- 复杂度分析

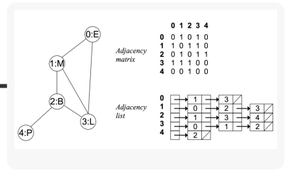
图graph

图的属性

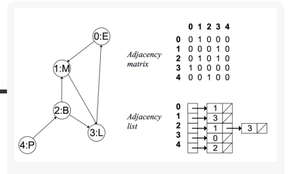


图的表示与分类

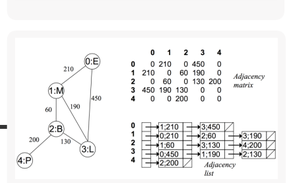
1. 无向无权图



2. 有向无权图



3. 无向有权图



4. 有向有权图 — 在上面的基础上有了方向

DFS代码

```
visited = set()  # 和树中的DFS最大区别

def dfs(node, visited):
    if node in visited: # terminator
        # already visited
        return

    visited.add(node)

    # process current node here.
    ...
    for next_node in node.children():
        if not next_node in visited:
            dfs(next_node, visited)
```

BFS代码

```
def BFS(graph, start, end):

    queue = []
    queue.append([start])

    visited = set() # 和数中的BFS的最大区别

    while queue:
        node = queue.pop()
        visited.add(node)

        process(node)
        nodes = generate_related_nodes(node)
        queue.push(nodes)
```

堆

堆 Heap

Heap: 可以迅速找到一堆数中的最大或者最小值的数据结构。

将根节点最大的堆叫做大顶堆或大根堆，根节点最小的堆叫做小顶堆或小根堆。常见的堆有二叉堆、斐波那契堆等。

假设是大顶堆，则常见操作（API）：

find-max: O(1)
delete-max: O(logN)
insert (create): O(logN) or O(1)

不同实现的比较： https://en.wikipedia.org/wiki/Heap_(data_structure)

二叉堆

- 二叉堆性质

二叉堆性质

通过完全二叉树来实现（注意：不是二叉搜索树）；

二叉堆（大顶）它满足下列性质：

[性质一] 是一棵完全树。
[性质二] 树中任意节点的值总是 >= 其子节点的值；

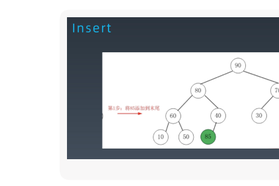
- 实现细节

二叉堆实现细节

1. 二叉堆一般都通过 “数组” 来实现
2. 假设"第一个元素"在数组中的索引为 0 的话，则父节点和子节点的位置关系如下：

(01) 索引为i的左孩子的索引是 (2*i+1);
(02) 索引为i的右孩子的索引是 (2*i+2);
(03) 索引为i的父结点的索引是 floor((i-1)/2);

- 对于一维数组



- 二叉堆

二叉堆

0. 根节点（顶堆元素）是：a[0]

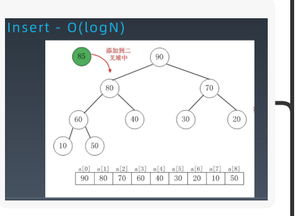
1. 索引为i的左孩子的索引是 (2*i+1);
2. 索引为i的右孩子的索引是 (2*i+2);
3. 索引为i的父结点的索引是 floor((i-1)/2);

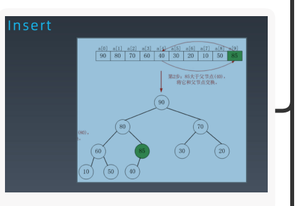
- Insert

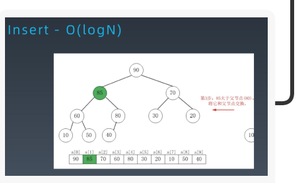
Insert 插入操作

1. 新元素一律先插入到堆的尾部
2. 依次向上调整整个堆的结构（一直到根即可）

HeapifyUp





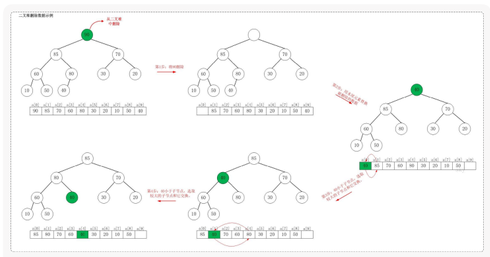


- Delete

Delete Max 删除堆顶操作

1. 将堆尾元素替换到顶部（即对顶被替代删除掉）
2. 依次从根部向下调整整个堆的结构（一直到堆尾即可）

HeapifyDown



- 其他 — java 里面实现 — 用prioityQueue实现的