堆：Heap　　二叉堆：Binary Heap

堆 Heap

特性：

Heap: 可以迅速找到一堆数中的最大或者(or)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)

堆的实现有很多种，

不要默认为二叉堆

相对比较容易实现

但时间效率较差

(与二叉搜索树没关系)

二叉堆性质

通过完全二叉树来实现（注意：不是二叉搜索树）；

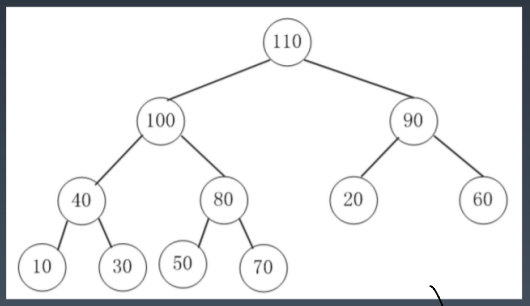
二叉堆（大顶）它满足下列性质：

[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；

它的根和每一级结点都是满的
(除了最下面一层叶子可能不满，
其他的上面都是满的)

For example:

二叉堆



("性质二"可以保证：它的根结点肯定是最大的结点。

(访问最大值
返回根结点值)

二叉堆实现细节

1. 二叉堆一般都通过“数组”(一维)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);

二叉堆

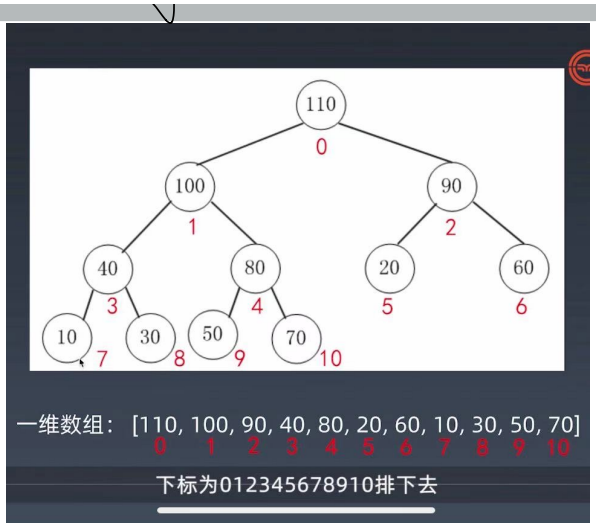
0. 根节点（顶堆元素）是：a[0]

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);

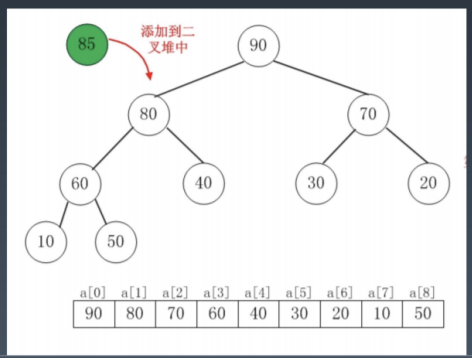
Insert 插入操作

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）

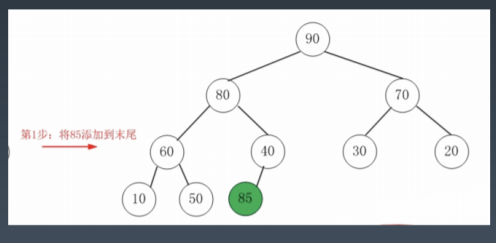
HeapifyUp



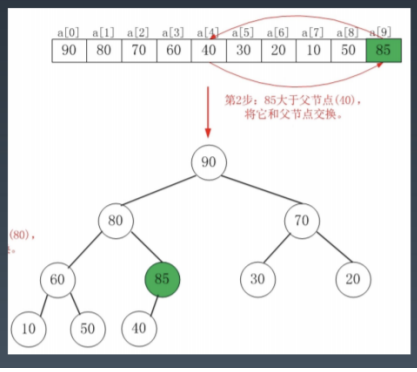
Insert - O(logN)



Insert

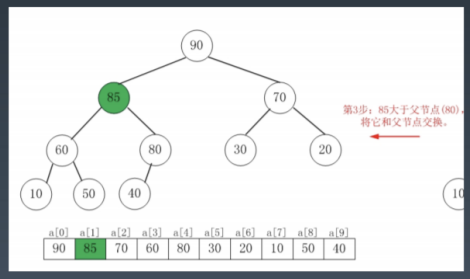


Insert



Insert O(logn)

Insert - O(logN)

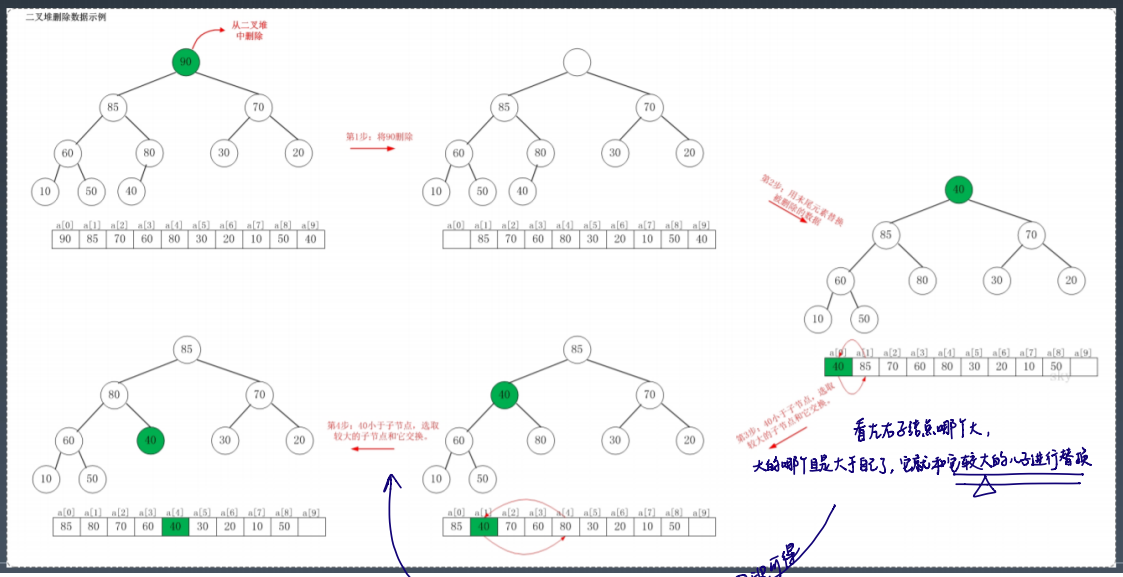


Delete Max 删除堆顶操作

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

Delete Max - O(logN)



堆：一个抽象的数据结构

注意：二叉堆是堆（优先队列 priority_queue）的一种常见且简单的实现；但是并不是最优的实现。

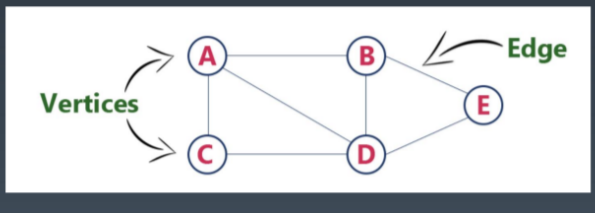
https://en.wikipedia.org/wiki/Heap_(data_structure)

Operation	find-min	delete-min	insert	decrease-key	meld
Binary[8]	$\Theta(1)$	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$	$\Theta(n)$
Leftist	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$
Binomial[8][9]	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$[b]	$\Theta(\log n)$	$O(\log n)$[c]
Fibonacci[8][10]	$\Theta(1)$	$O(\log n)$[b]	$\Theta(1)$	$\Theta(1)$[b]	$\Theta(1)$
Pairing[11]	$\Theta(1)$	$O(\log n)$[b]	$\Theta(1)$	$o(\log n)$[b][d]	$\Theta(1)$
Brodal[14][e]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Rank-pairing[16]	$\Theta(1)$	$O(\log n)$[b]	$\Theta(1)$	$\Theta(1)$[b]	$\Theta(1)$
Strict Fibonacci[17]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2-3 heap[18]	$O(\log n)$	$O(\log n)$[b]	$O(\log n)$[b]	$\Theta(1)$	?

图 { 图的属性&分类
关于图的相关算法

图
Graph

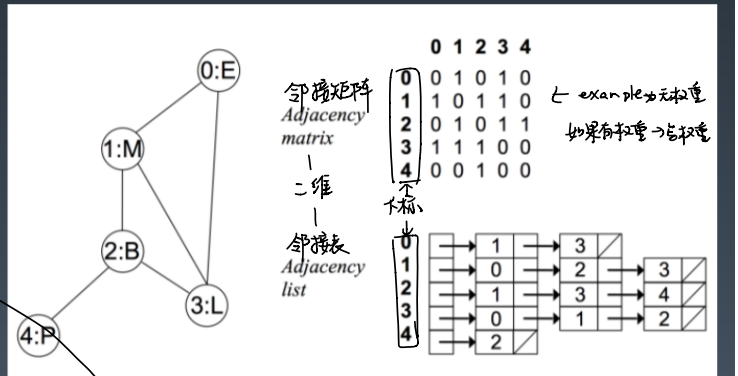
定义：有点，有边



图的属性

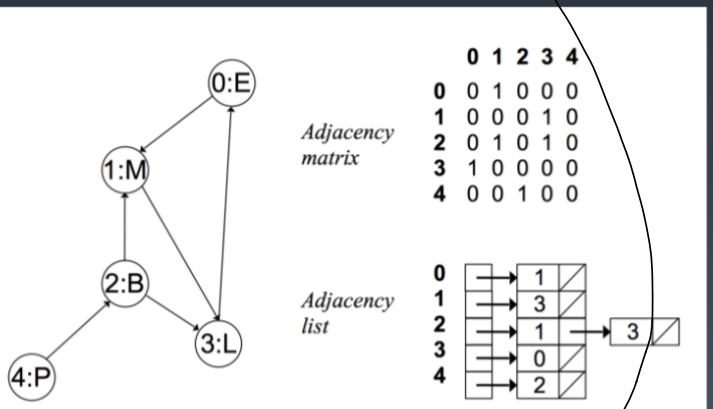
- Graph(V, E)
- V - vertex: 点
  1. 度 - 入度和出度 (点连了多少边)
  2. 点与点之间：连通与否
- E - edge: 边
  1. 有向和无向（单行线）
  2. 权重（边长）/损耗

图：无向无权图

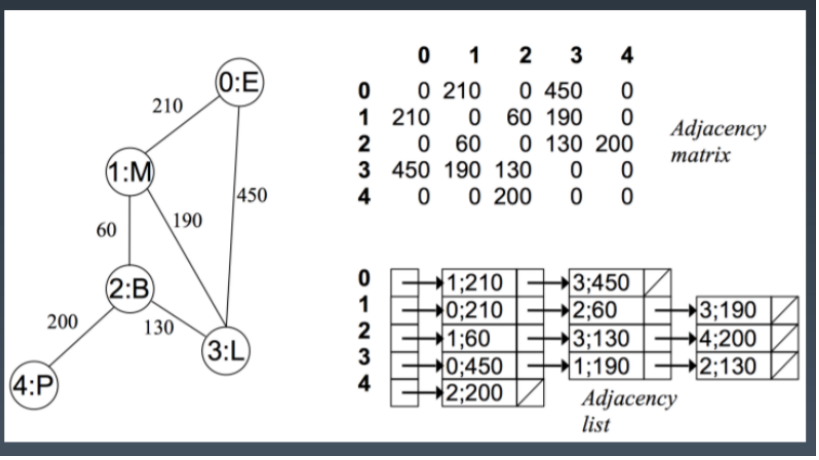


图的表示和分类
有向有权图

图：有向无权图



图：无向有权图



基于图的相关算法：BFS

DFS

> 在图里，不要忘记加visited的集合

DFS → Python

```
visited = set()  # 和树中的DFS最大区别
```

'树'中不会可以保证它没有所谓的环路

访问它的点可以永远不会重复

图可能会有重复

代码模板

一定要记住背

BFS → Python：

```
visited = set()  # 和数中的BFS的最大区别
```