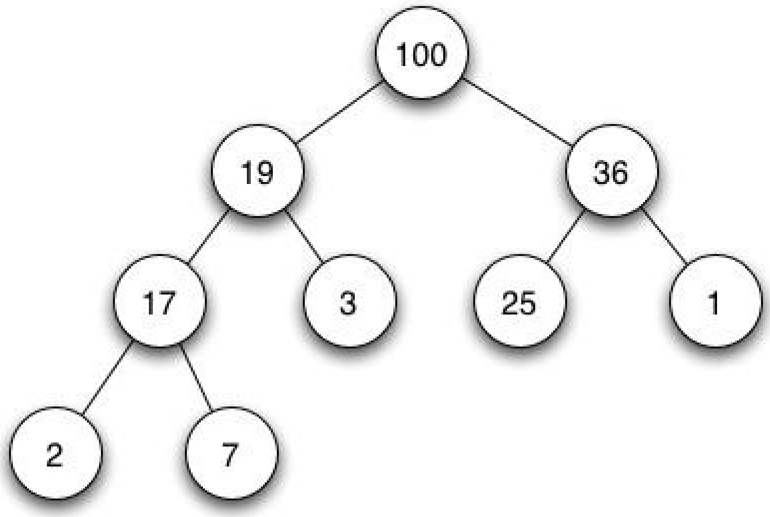
堆

前置知识需要:二叉树和完全二叉树,部分向上下取整的公式

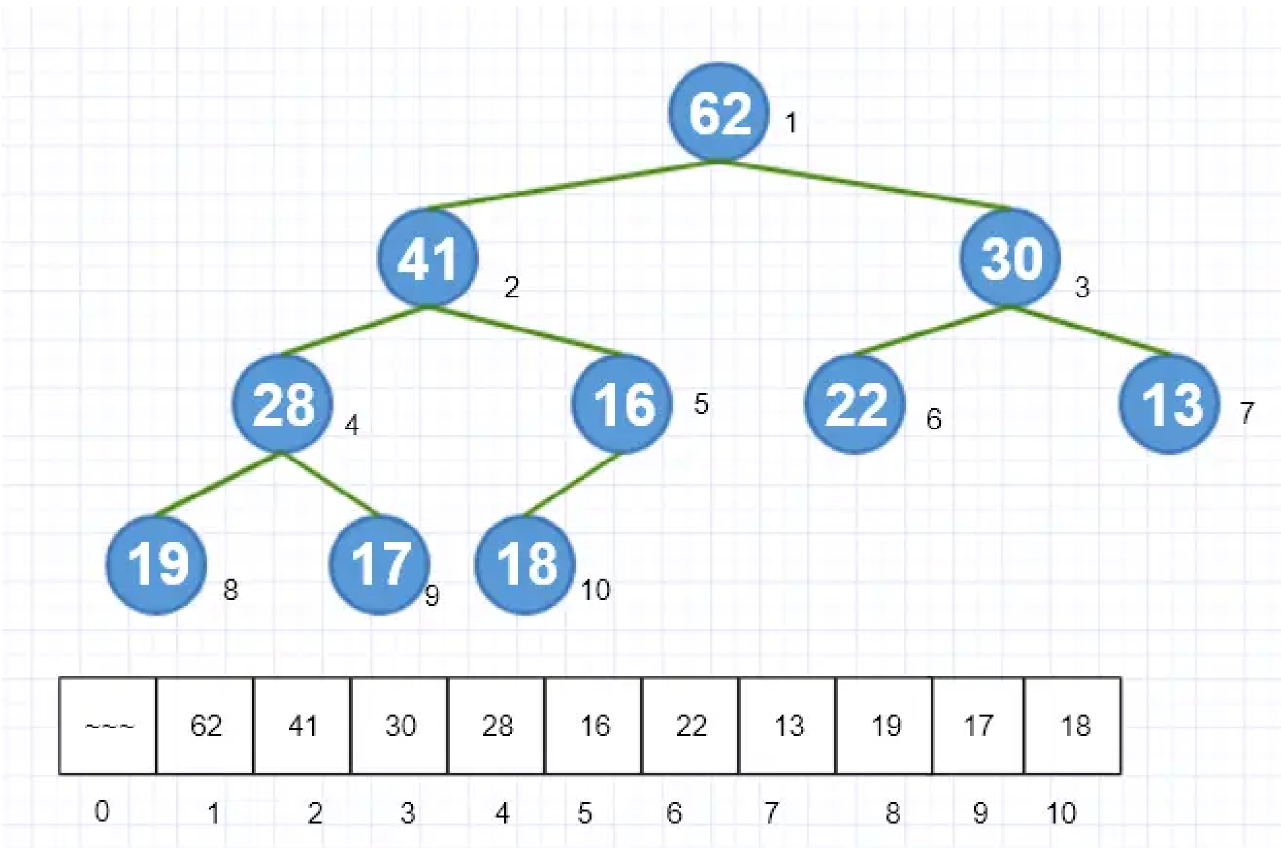
1. floor(n/2)+ceiling(n/2)=n
2. ceiling(ceiling(n/a)/b)=ceiling(n/(a*b))

堆的基础概念

1. 对于一个最大堆,他的左右节点的值都比它的值要小,最小堆反之



2. 得益于完全二叉树的性质,可以用数组来表示一个堆,节省了空间



3. 对于任意一个节点的下标为i,他的左儿子下标为2i,右儿子2i+1,父亲是i/2

证明题

1. 对于一个有n个节点的(最大/最小)二叉堆,它的高度是logn
2. 叶节点的下标是floor(n/2)+1 ... n

堆的操作

下滤

1. 堆的出堆和建堆时都是使用下滤操作
2. 对于某个节点,如果它的左右子节点都符合(最大堆/最小堆)的性质,在进行下滤操作后
3. 这个节点将变成一个最大堆/最小堆。

伪代码如下:

```
def MaxHeapOut(i=1):
    return_value=heap[1]
    heap[1]=heap[size--]
    while (l<=size)
    {
        l=2*i
        r=2*i+1
        find maxone in leftson and rightson
        if (maxone<heap[i])
        exchange(i,maxone对应的下标)
        i=maxone对应的下标
    }
    return return_value
```

上滤

1. 上滤通常使用在插入操作,对于插入操作,我们让当前子节点与父节点比较,如果不符合规律就互换
2. 如果交换,就继续往上看能不能交换,反之上滤完毕

伪代码如下

```
def MaxHeapInsert(value)
    heap[++size]=value
    i=size
    while (i/2):
    {
        if (heap[i]>heap[i/2]):
            swap(heap[i],heap[i/2])
            i/=2
            continue
```

```
        else:
            break
    }
```

证明题

1. 上滤和下滤的时间复杂度O(logn)
2. 插入建堆的复杂度分析

线性建堆

我们已经学习了下滤,了解到下滤的本质操作是对一个左右儿子都符合堆的性质的节点时,对这个节点进行下滤操作,就可以使这一整个成为堆。那我们意识到,如果对一个啥也不是但是是一个完全二叉树来说,高度为0的节点都是一个最大/最小堆,对于高度为1的节点,它的左右节点是堆,那么进行下滤就可以让这个高度为1的节点成为一个堆,那么逐层递推到高度为n,一个堆就建完了

```
def BuildMaxHeap(int arr[],int length):
    size=length
    for i=length/2 downto 1:
        Downfloat(i)
```

分析复杂度前的前置证明

证明:对于任一包含n个节点的堆中,至多有ceiling(n/2^h+1)个高度为h的节点

对于线性建堆的复杂度分析

堆排序

~~学习了上滤下滤线性建堆后,那么堆排序就很简单了~~ 分析之后我们知道建堆花费O(n),紧接着逐层出堆即可得到一个有序的数列

堆拓展

题型一:第k大/小问题

给你一个一直会动态增加元素的数组,增加元素的过程中我将会询问你第k小/大的元素是什么?

1. 暴力解决
2. 堆解决

题型二:动态的第k大/小的问题

给你一个一直会动态增加元素的数组,增加元素过程中我将询问当前数组的中位数是什么?

1. 仍然是暴力
2. 对顶堆解决