链滴

堆排序 go 实现

作者：Gakkiyomi2019

原文链接：https://ld246.com/article/1602314466628

来源网站：链滴

堆

想实现堆排序，必须了解数据结构 ---堆。

那么什么是堆呢？ 在我看来堆就是一颗完全二叉树的数组形式。

根据其节点的大小，又分为 **大顶堆**和**小顶堆。**

堆虽然是树但是不用存子节点的指针，所以内存占用较小。堆主要用来排序，用来搜索性能较低。

他们的数组索引 就是树的层次遍历次序。

也可以得到获取其左右子节点的公式如下:

```
大顶堆：arr[i] >= arr[2i+1] && arr[i] >= arr[2i+2]
小顶堆：arr[i] <= arr[2i+1] && arr[i] <= arr[2i+2]
```

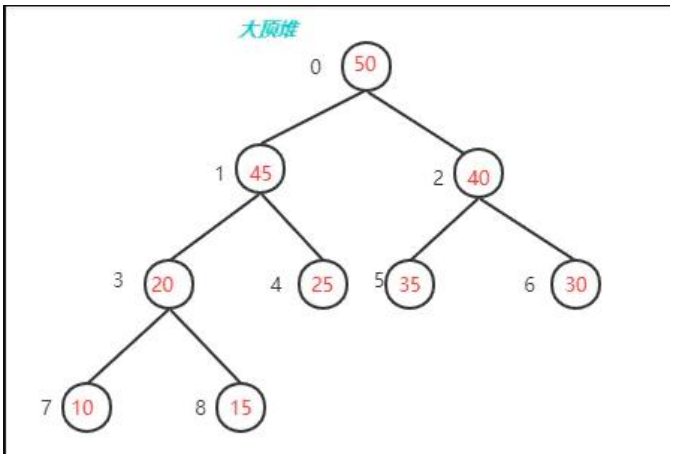
根据这个公式 我们可以先用代码实现下堆的特性：

```
Heap struct {
        Items []int
    }

func (hp *Heap) GetLeftIndex(parentIndex int) int {
    return 2*parentIndex + 1
}

func (hp *Heap) GetRightIndex(parentIndex int) int {
    return 2*parentIndex + 2
}

func (hp *Heap) GetParentIndex(index int) int {
```

```
    return (index - 1) / 2
}

func (hp *Heap) GetLeft(parentIndex int) int {
    return hp.Items[hp.GetLeftIndex(parentIndex)]
}

func (hp *Heap) GetRight(parentIndex int) int {
    return hp.Items[hp.GetRightIndex(parentIndex)]
}

func (hp *Heap) GetParent(index int) int {
    return hp.Items[hp.GetParentIndex(index)]
}

func (hp *Heap) HasLeft(index int) bool {
    return hp.GetLeftIndex(index) < len(hp.Items)
}

func (hp *Heap) HasRight(index int) bool {
    return hp.GetRightIndex(index) < len(hp.Items)
}

func (hp *Heap) HasParent(index int) bool {
    return hp.GetParentIndex(index) >= 0
}

func (hp *Heap) Swap(index1, index2 int) {
    hp.Items[index1], hp.Items[index2] = hp.Items[index2], hp.Items[index1]
}
```

大顶堆

特性： 其中父节点一定大于左右子节点。



在堆的基础上保证其父节点一定大于左右子节点即可，并且在插入元素和删除元素后需要调整新并仍持为正确的大顶堆结构。

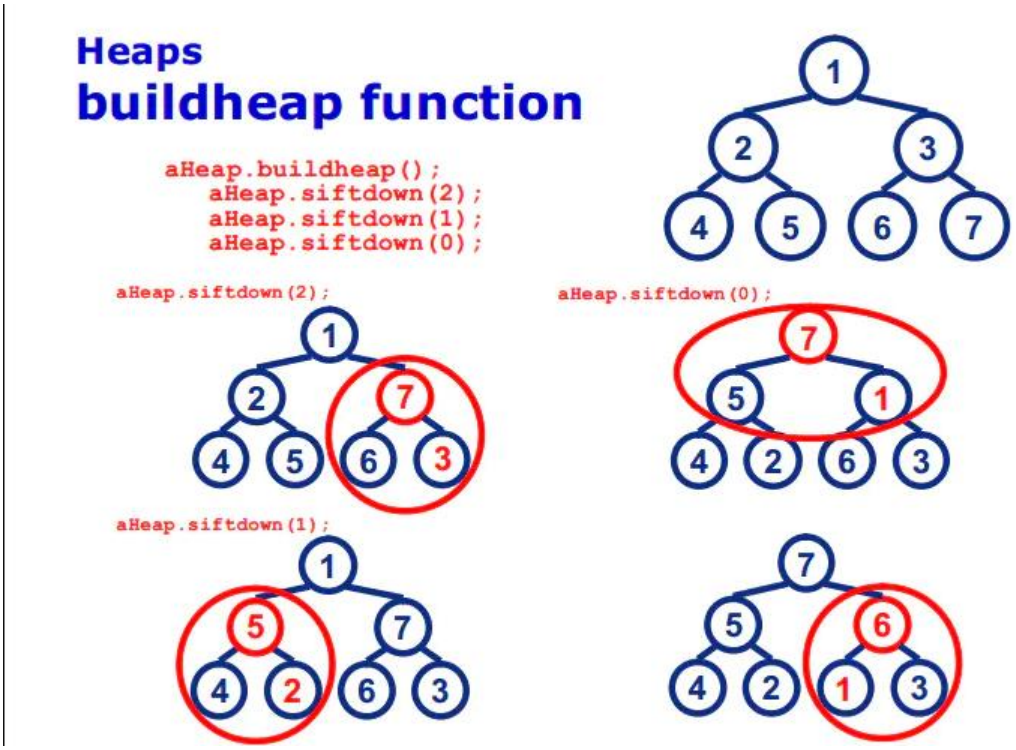
```
type MaxHeap struct {
    *Heap
```

```
}

func NewMaxHeap(source []int) *MaxHeap {
    h := &MaxHeap{
        &Heap{
            Items: source,
        },
    }

    if len(h.Items) > 0 {
        h.buildMaxHeap()
    }

    return h
}

//对于叶子节点，不用调整次序，根据满二叉树的性质，叶子节点比内部节点的个数多1.所以i=n/2 -1
，不用从n开始。
func (h *MaxHeap) buildMaxHeap() {
    for i := len(h.Items)/2 - 1; i >= 0; i-- {
        h.shiftDown(i)
    }
}

func (h *MaxHeap) Insert(item int) *MaxHeap {
    h.Items = append(h.Items, item)
    h.shiftUp(len(h.Items) - 1)
    return h
}

func (h *MaxHeap) ExtractMax() int {
    if len(h.Items) == 0 {
        logs.Error("No items in the heap")
    }
    minItem := h.Items[0]

    h.Items[0] = h.Items[len(h.Items)-1]

    h.Items = h.Items[:len(h.Items)-1]

    h.shiftDown(0)

    return minItem
}

func (h *MaxHeap) shiftUp(index int) {
    for h.HasParent(index) && h.GetParent(index) < h.Items[index] {
        h.Swap(h.GetParentIndex(index), index)
        index = h.GetParentIndex(index)
    }
}

func (h *MaxHeap) shiftDown(index int) {
```

```
    //如果当前index存在左或者右节点并且大于当前节点的值则需要交换
    for (h.HasLeft(index) && h.Items[index] < h.GetLeft(index)) || (h.HasRight(index) && h.Items
index] < h.GetRight(index)) {
        //如果左右节点都大于父节点
        if (h.HasLeft(index) && h.Items[index] < h.GetLeft(index)) && (h.HasRight(index) && h.It
ms[index] < h.GetRight(index)) {
            //找到较大的一个的进行交换
            if h.GetLeft(index) > h.GetRight(index) {
                h.Swap(index, h.GetLeftIndex(index))
                index = h.GetLeftIndex(index)
            } else {
                h.Swap(index, h.GetRightIndex(index))
                index = h.GetRightIndex(index)
            }
        } else if h.HasLeft(index) && h.Items[index] < h.GetLeft(index) { //只有左节点大于父节点
            h.Swap(index, h.GetLeftIndex(index))
            index = h.GetLeftIndex(index)
        } else { //只有右节点大于父节点
            h.Swap(index, h.GetRightIndex(index))
            index = h.GetRightIndex(index)
        }
    }
}
```

buildMaxHeap

```
//对于叶子节点，不用调整次序，根据满二叉树的性质，叶子节点比内部节点的个数多1.所以i=n/2 -1
，不用从n开始。
func (h *MaxHeap) buildMaxHeap() {
    for i := len(h.Items)/2 - 1; i >= 0; i-- {
        h.shiftDown(i)
    }
}
```

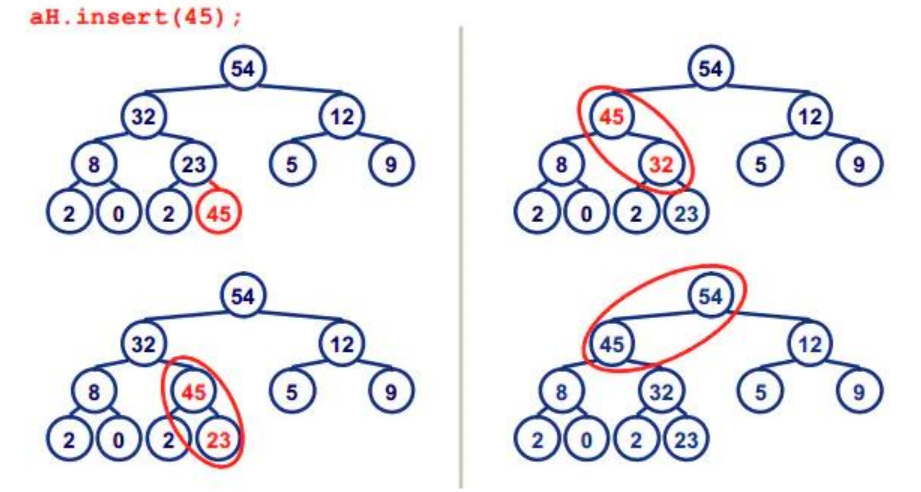


原文链接：堆排序 go 实现

insert

```
func (h *MaxHeap) Insert(item int) *MaxHeap {
    h.Items = append(h.Items, item)
    h.shiftUp(len(h.Items) - 1)
    return h
}
```

每次插入节点都是放在追加到数组最后，所以需要判断大小往上冒。

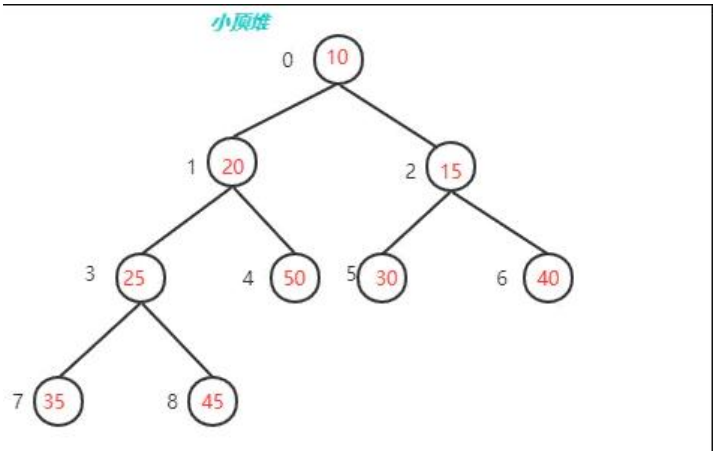


ExtractMax

取出第一个值，然后重新构建大顶堆

小顶堆

特性： 其中父节点一定小于于左右子节点。



小顶堆的实现方式与大顶堆完全一样，只要改下判断条件即可。

源码： 堆

堆排序

实现了大小顶堆之后，我们只要拿到最大值或者最小值就可以升序或者降序排列。

```
func HeapSort(source []int, asc bool) []int {

    result := []int{}

    if asc {

        minHeap := structure.NewMinHeap(source)
        for range source {
            // 也可以使用栈或者队列
            //result = array.Array.InsertAtIndexByIntArray(result, maxHeap.ExtractMax(), 0)
            result = append(result, minHeap.ExtractMin())
        }

    } else {
        maxHeap := structure.NewMaxHeap(source)
        for range source {
            result = append(result, maxHeap.ExtractMax())
        }

    }
    return result
}
```

更多排序算法源码：排序算法