第6章 堆排序（二叉堆）

2020年10月20日　　11:01

6.1-6.4 堆排序及排序

1. 堆及堆的表示

   堆：一颗完全二叉树

   存储结构：数组A，如Fig 6.1

   2个属性：A.length，A.heapsize

2. 堆性质

   子节点与父节点的下标关系：

   i的孩子节点：$left(i)=2i, right(i)=2i+1$

   i的父节点：$parent(i)=\left\lfloor \frac{i}{2} \right\rfloor$

   n个节点的堆，其叶子节点：$\left\lfloor \frac{n}{2} \right\rfloor+1, \left\lfloor \frac{n}{2} \right\rfloor+2, \dots, n$

3. 大/小根堆

   $\forall\, node\ i\ \in T: A[parent(i)] \geq A[i]$(大根堆)

   $\forall\, node\ i\ \in T: A[parent(i)] \leq A[i]$(小根堆)

4. 整堆

   （1）算法

   前提和目标

   设节点i，其left(i)和right(i)均为大根堆，但节点i不满足大根堆要求

   目标将以节点i为根的子树整为大根堆

```
MAX − HEAPIFY(A, i)
    l = left(i)
    r = right(i)
    if l ≤ A.heapsize and A[l] > A[i]：
        largest = l
    else
        largest = i
    if r ≤ A.heapsize and A[r] > A[largest]：
        largest = r
    if largest ≠ i：
        exchange A[i] with A[largest]
        MAX − HEAPIFY(A, largest)
```

   （2）计算示例：Fig 6.2

   （3）时间：$O(\log n)$ $//\because T(n) \leq T\left(\frac{2}{3}n\right)+\theta(1)$

5. 建堆

(1) 算法

从$\left\lfloor\frac{A.length}{2}\right\rfloor$处开始整堆，至$\left\lfloor\frac{A.length}{2}\right\rfloor - 1, \dots,$
直至1（树根）

$BUILD - MAX - HEAP(A)$

$\quad A.heapsize = A.length$

$\quad for\ i = \frac{\lfloor A.length \rfloor}{2}\ downto\ 1$：

$\quad\quad MAX - HEAPIFY(A, i)$

(2) 计算示例：Fig 6.3

(3) 时间：$O(n)$ //一般分析为$O(n \log n)$,但不紧

6. 堆排序

(1) 算法

$HEAPSORT(A)$

$\quad BUILD - MAX - HEAP(A)$

$\quad for\ i = A.length\ downto\ 2$：

$\quad\quad exchange\ A[1]\ with\ A[i]$

$\quad\quad A.heapsize = A.heapsize - 1$

$\quad\quad MAX - HEAPIFY(A, 1)$

(2) 计算示例：Fig 6.4

(3) 时间：$O(n \log n)$(与快速排序相比，其常系数比较大)

6.5 优先队列

FIFO队列：先进先出

优先队列：按重要性（优先级）次序出队

优先队列支持的操作：

①$Insert(S, x)$：将x插入到S中(入队)

②$Maximum(S)$：返回最大key元素

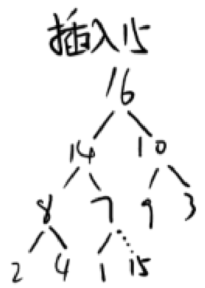
③$ExtractMax(S)$：删最大元素并返回其值

④$HeapIncrease(S, x, k)$：将x的值增至k

①②③④是API库，运行时

1. 插入（入队）

   idea：

   算法步骤：

   加入到最后位置

   自底向上进行整堆

   伪代码：

   $MaxHeapInsert(A, key)$
   $\quad A.heapsize \mathrel{+}= 1$
   $\quad i = A.heapsize$
   $\quad while\ i > 1\ and\ A[parent(i)] < key$：
   $\quad\quad A[i] = A[parent(i)]$
   $\quad\quad i = parent(i)$
   $\quad A[i] = key$

   时间分析：$O(\log n)$ 最坏情况

2. 取最大关键字

   …
   $return\ A[1]$
   …

   时间分析：$O(1)$

3. 删除（出队）

   算法步骤：

   a. 取堆顶$A[i]$
   b. 将$A[A.heapsize] \to A[1]$
   c. $A.heapsize --$
   d. 对$A[1]$进行整堆

   伪代码：

   $HeapExtractMax(A)$
   $\quad if\ A.heapsize < 1$：
   $\quad\quad error(\text{underflow})$
   $\quad max = A[1]$
   $\quad A[1] = A[A.heapsize]$
   $\quad A.heapsize --$
   $\quad MaxHeapify(A, 1)$ //调用整堆函数
   $\quad return$ max

   时间分析：$O(\log n)$

4. 增值（修改）

伪代码：

$HeapIncreaseKey(A, i, key)$

$\quad if\ key \le A[i]$：

$\quad\quad A[i] = key$

$\quad\quad Heapify(A, i)$ （调用整堆函数）

$\quad else$

$\quad\quad while\ i > 1\ and\ A[parent(i)] < key$：

$\quad\quad\quad A[i] = A[parent(i)]$

$\quad\quad\quad i = parent(i)$

$\quad\quad A[i] = key$

时间分析：$O(\log n)$

注：用增值操作代替入队操作

a. $A.heapsize++$
b. $A[A.heapsize] = -\infty$
c. $HeapIncreaseKey(A, A.heapsize, key)$