Lab 4 实验报告

1 d 叉堆的表示方法

1. d 叉堆的元素和二叉堆一样，也有 d 叉树形式和数组形式两种表示方法。每一个节点对应数组中的一个元素。d 叉树从根节点开始，从上到下从左到右，一一与数组中的元素对应。

2. 在本程序中，树的根节点在数组中的下标设定为 0。

3. 以下计算第 i 行第 j 列的元素在数组中的下标。设其下标为 n(i,j). 那么有

$$\begin{aligned} n(i,j) &= 1 + d + d^2 + \cdots + d^{i-2} + j - 1 \\ &= \frac{1 \times \ (1 - d^{i-1})}{1-d} + j - 1 \\ &= \frac{d^{i-1} - 1}{d-1} + j - 1 \\ &= \frac{d^{i-1} - d}{d-1} + j \end{aligned} \tag{1}$$

4. 以下计算其父节点在数组中的下标，其父节点应当在第 i-1 行第 $\lceil j/d \rceil$ 个。设其下标为 p(i,j)，在计算过程中用 (1) 式结果代入，那么有

$$\begin{aligned} p(i,j) &= 1 + d + d^2 + \cdots + d^{i-3} + \lceil j/d \rceil - 1 \\ &= \frac{1 \times \ (1 - d^{i-2})}{1-d} + \lfloor \frac{j+d-1}{d} \rfloor - 1 \\ &= \frac{d^{i-1} - d}{d(d-1)} + \lfloor \frac{j+d-1}{d} \rfloor - 1 \\ &= \lfloor \frac{1}{d} \times \ (\frac{d^{i-1} - d}{d-1} + j + d - 1) \rfloor - 1 \\ &= \lfloor \frac{1}{d} \times \ (n(i,j) + d - 1) \rfloor - 1 \\ &= \lfloor \frac{1}{d} \times \ (n(i,j) - 1) \rfloor \end{aligned} \tag{2}$$

5. 以下计算其第 n 个孩子节点在数组中的下标，其第 n 个孩子节点应当在第 i+1 行第

$d\times\ (j{-}1)+n$ 个。设其下标为 c(i,j,n)，在计算过程中用 (1) 式结果代入，那么有

$$
\begin{aligned}
c(i,j,n) &= 1+d+d^2+\cdots+d^{i-1}+d(j{-}1)+n-1\\
&= \frac{1\times\ (1-d^i)}{1-d}+d(j{-}1)+n-1\\
&= \frac{d^i-1}{d-1}+d(j{-}1)+n-1\\
&= \frac{d^i-d}{d-1}+d(j{-}1)+n\\
&= \frac{d^i-d^2}{d-1}+d\times\ j+n\\
&= d\times\ n(i,j)+n
\end{aligned}
\tag{3}
$$

6. 于是，由 (2)(3) 递推式的结果，我们得到了 d 叉堆在数组中的表示方式，如算法 1 所示。

算法 1: 父节点和孩子节点的函数

```python
def parent(i, d):
        return (i-1) // d

def nthChild(i, n, d):
        return d*i + n
```

2 d 叉堆的高度

1. 假定现在要计算高度的 d 叉堆含有 n 个元素。
2. 首先来计算高度为 h 的 d 叉堆至少含有的元素个数。高度为 h 的 d 叉堆一共有 h+1 层，其中最后一层至少要有一个元素。

$$
\begin{aligned}
min(n) &= 1+d+d^2+\cdots+d^{h-1}+1\\
&= \frac{d^i-1}{d-1}+1
\end{aligned}
\tag{4}
$$

3. 然后计算高度为 h 的 d 叉堆至多含有的元素个数。高度为 h 的 d 叉堆一共有 h+1 层，其中最后一层叶子节点是满的。

$$
\begin{aligned}
max(n) &= 1+d+d^2+\cdots+d^{h-1}+d^h\\
&= \frac{d^{h+1}-1}{d-1}
\end{aligned}
\tag{5}
$$

4. 那么，由 (4)(5) 两式，我们将得到元素个数 n 的取值范围。

$$\frac{d^h - 1}{d - 1} + 1 \leq n \leq \frac{d^{h+1} - 1}{d - 1}$$

$$d^h + d - 2 \leq (d-1)n \leq \ d^{h+1} - 1$$

5. 由 d 叉树的实际意义，d>=2 且 d 是正整数，因此

$$d^h \leq (d-1)n < d^{h+1}$$

$$h = \lfloor \log_d((d-1)n) \rfloor$$

6. 所以 d 叉堆的高度 h 为 $\lfloor \log_d((d-1)n) \rfloor$。

3 extractMax() 函数实现及其运行时间分析

1. 要实现 extractMax() 函数，就需要调用 maxHeapify() 函数。所以先给出 maxHeapify() 函数的实现，并分析其运行时间。

算法 2: maxHeapify() 函数的实现

```
 def findLargest(self, i):
        largest = i
        for j in range(1, self.__d + 1):
                child = nthChild(i, j, self.__d)
                if(child < self.__size):
                        if(self.__heap[child] > self.__heap[largest]):
                                largest = child
                else:
                        break
        return largest

def maxHeapify(self, i):
        largest = self.findLargest(i)
        if(i != largest):
                self.__heap[i], self.__heap[largest] = \
                        self.__heap[largest], self.__heap[i]
                self.maxHeapify(largest)
```

2. 每次调用 findLargest() 函数时，预设指定节点本身为 largest，与其最多 d 个孩子进行最多 d 次比较。在 maxHeapify() 函数中调用 findLargest() 函数，如果得到最大的是指定节点的一个孩子节点，就交换它们的位置，并在原孩子节点处再次调用 maxHeapify() 函数，最多直到叶子节点不调用，最多交换和调用次数为 d 叉堆的高度 h。

3. 故 maxHeapify() 函数的运行时间为

$$\begin{aligned} O(dh+h) &= O(h(d+1)) \\ &= O(\lfloor \log_d((d-1)n) \rfloor (d+1)) \\ &= O(dlog_d n) \\ &= O(log_d n) \end{aligned} \tag{6}$$

4. 再给出 extractMax() 函数的实现。

算法 3: extractMax() 函数的实现

```
def extractMax(self):
        if(self.__size < 1):
                print("Error! Heap underflow!")
                return
        maxn = self.__heap[0]
        self.__size = self.__size - 1
        t = self.__heap.pop()
        if(self.__size > 0):
                self.__heap[0] = t
        self.maxHeapify(0)
        return maxn
```

5. extractMax() 函数会删去并返回 d 叉堆数组中第一个数，也就是最大的数；size 减少 1；把最后一个数（如果有的话）移到第一个数的位置；再对现在第一个数调用 maxHeapify() 函数。由于其他操作都在常数时间内完成，所以 extractMax() 函数的运行时间取决于 maxHeapify() 函数的运行时间，由 (6) 式得答案即 $O(log_d n)$。

4 increaseKey() 函数实现及其运行时间分析

1. 由于要实现 insert() 函数，就需要调用 increaseKey() 函数。所以先给出 increaseKey() 函数的实现，并讨论其运行时间。

算法 4: extractMax() 函数的实现

```
def increaseKey(self, i, key):
        if(self.__heap[i] < key):
                self.__heap[i] = key
                while(i > 0):
                        pa = parent(i, self.__d)
                        if(self.__heap[pa] < self.__heap[i]):
                                self.__heap[i], self.__heap[pa] = \
                                        self.__heap[pa], self.__heap[i]
                                i = pa
                        else:
                                break
```

2. increaseKey() 函数将 key 值和当前节点本身的值中较大者赋给当前节点。如果当前节点的值增大，就与其父节点比较。如果当前节点更大，就交换两者位置并继续与上一个节点比较，最多直到根节点不比较。则最多交换和比较次数为 d 叉堆的高度 h。

3. 故 increaseKey() 函数的运行时间为

$$
\begin{aligned}
O(1+2h) &= O(h) \\
&= O(\lfloor \log_d((d-1)n) \rfloor) \\
&= O(log_d n)
\end{aligned} \tag{7}
$$

5 insert() 函数实现及其运行时间分析

1. 先给出 insert() 函数的实现，再讨论其运行时间。

算法 5: insert() 函数的实现

```
def insert(self, key):
        self.__heap.append(-99999999)
        self.__size = self.__size + 1
        self.increaseKey(self.__size - 1, key)
```

2. insert() 函数会在 d 叉堆数组后面加上一个无限小的数；size 增加 1；再对新加入的数调用 increaseKey() 函数。由于其他操作都在常数时间内完成，所以 insert() 函数的运行时间取决于 increaseKey() 函数的运行时间，由 (7) 式可知答案即 $O(log_d n)$。