数据结构第 4 次上机实验报告

林子开

2023 年 9 月 30 日

目录

1 The representation of a d-ary heap in an array

Assume the array as A. Let the first element A[1] be the root node. Then we have the relation:

The index of the parent of A[i], is $\lfloor \frac{i+(d-2)}{d} \rfloor$, and the range of the indexes of the children of A[i] are $[i \times d - (d-2), i \times d + 1]$.

2 The height of a d-ary heap

The height of a d-ary heap should satisify:

$$\begin{aligned} & \frac{d^{h-1}-1}{d-1} < n \leq \frac{d^h-1}{d-1} \\ \Rightarrow\ & d^{h-1} < (d-1)n+1 \leq d^h \\ \Rightarrow\ & h-1 < \log_d[(d-1)n+1] \leq h \end{aligned}$$

Then we can conclude:

$$h = \lceil \log_d[(d-1)n+1] \rceil = O(\log_d n)$$

3 The implementation of EXTRA-MAX function

Listing 1: The implementation of EXTRA-MAX

```
    def extra_max(self):
        if self.currentSize < 1:
            print('错误：heap已经空的！')
            return
        max_item = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize -= 1
        self.heapify(1)
        print('弹出最大元素！最大元素是：',max_item)
        return max_item

    def heapify(self,i):
        child_range = self.childs(i)
        temp = None
        largest = i
        for k in child_range:
            if k <=self.currentSize and self.heapList[k] > self.heapList[i]:
                largest = k
        if largest != i:
            temp = self.heapList[i]
            self.heapList[i] = self.heapList[largest]
            self.heapList[largest] = temp
            self.heapify(largest)
```

The running time is determined by the process of heapyfying A[1]. **Since we need to compared a parent node with its d children in every iteration, therefore the running time is** $O(dh) = O(d \log_d n)$**.**

4 The implementation of INSERT function

Listing 2: The implementation of INSERT

```
    def insert(self, key):
        self.currentSize += 1
        self.heapList.append(None)
        self.increase_key(self.currentSize, key)
```

The running time is determined by the process of increasing the key of A[heapSize], which is also $O(h) = O(\log_d n)$.

5 The implementation of INCREASE-KEY function

Listing 3: The implementation of INCREASE-KEY

```
    def increase_key(self, i, key):
        if i > self.currentSize:
            print('错误：下标已经超出数组范围！')
        if not self.heapList[i]:  # if None
            self.heapList[i] = key
```

```
            # print(self.heapList)
            while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
                temp = self.heapList[i]
                self.heapList[i] = self.heapList[self.parent(i)]
                self.heapList[self.parent(i)] = temp
                i = self.parent(i)
                # print(self.heapList)
            return
        elif key >= self.heapList[i]:
            self.heapList[i] = key
            while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
                temp = self.heapList[i]
                self.heapList[i] = self.heapList[self.parent(i)]
                self.heapList[self.parent(i)] = temp
            return
        elif key < self.heapList[i]:
            print('错误：输入的key值比原有的数值小！')
            print('当前heap的情况为',self.heapList)
            print('key=',key)
            print('position=',i)
```

When increasing the key of A[i] to k, we need to find the proper position. The worst case occurs when we need to update A[i] from the leaf to the root, where the length of path is the height of heap. **Since we only need to compare a child with its parent node for only once, therefore the running time of INCREASE-KEY is** $O(h) = O(\log_d n)$**.**

6 Constructing a 3-ary heap

After constructing the 3-ary heap and inserting a sequence ranging from 1 to 30, the heap is like:

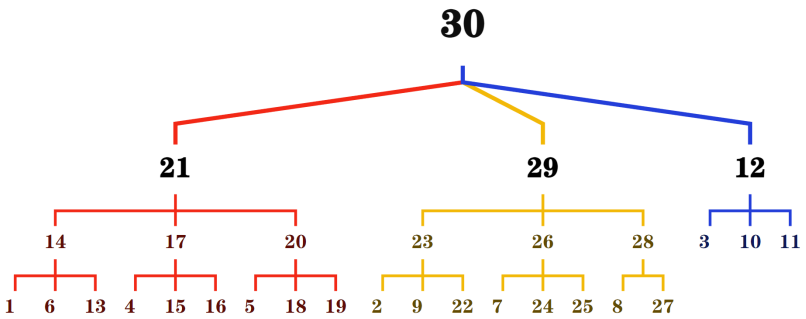

图 1: The heap after inserting a sequence ranging from 1 to 30

After performing the operation EXTRA-MAX, **the function return 30**. And the heap is as follows:

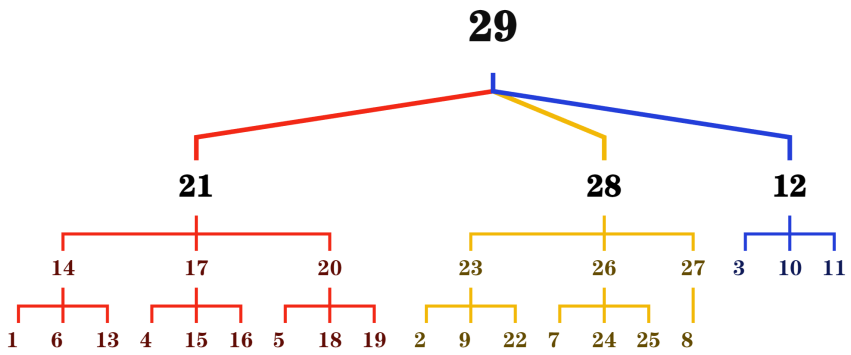

图 2: The heap after performing EXTRA-MAX

After performing the operation of INCREASE-KEY(A, 10, 28), the heap looks like:

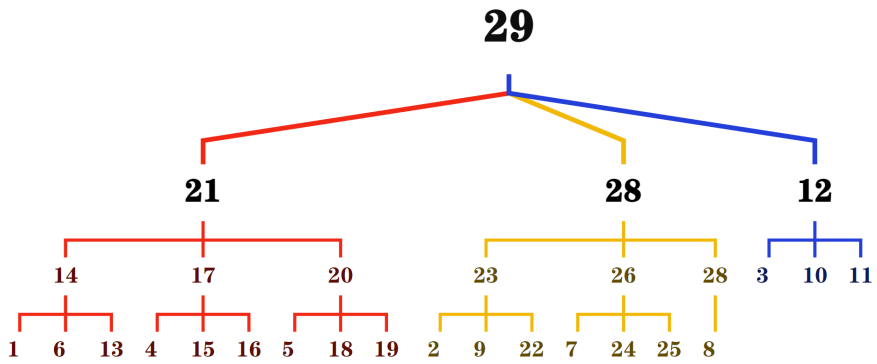

图 3: The heap after performing INCREASE-KEY(A, 10, 28)

Appendix: all of my python codes

Listing 4: All of my python codes

```
from math import *

class d_ary_heap:
    def __init__ (self,d):
        self.heapList = [0]
        self.d_factor = d
        self.currentSize = 0

    def show_heap(self):
        d = self.d_factor
        print('\n', str(d),'叉堆如下\n')
        count = 1
        for level in  range(ceil(log(self.currentSize, d))):
            for _ in  range(d**level):
                if count > self.currentSize:
                    print('\n')
```

```
                    return
                else:
                    print(self.heapList[count],end=' ')
                    count += 1
            print('\n')




    def childs(self,i):
        temp = i * self.d_factor
        return  tuple( range(temp - (self.d_factor - 2), temp + 1 + 1))

    def parent(self,i):
        return (i + (self.d_factor-2))//self.d_factor

    def extra_max(self):
        if self.currentSize < 1:
            print('错误：heap已经空的！')
            return
        max_item = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize -= 1
        self.heapify(1)
        print('弹出最大元素！最大元素是：',max_item)
        return max_item

    def heapify(self,i):
        child_range = self.childs(i)
        temp = None
        largest = i
        for k in child_range:
            if k <=self.currentSize and self.heapList[k] > self.heapList[i]:
                largest = k
        if largest != i:
            temp = self.heapList[i]
            self.heapList[i] = self.heapList[largest]
            self.heapList[largest] = temp
            self.heapify(largest)

    def insert(self, key):
        self.currentSize += 1
        self.heapList.append(None)
        self.increase_key(self.currentSize, key)

    def increase_key(self, i, key):
        if i > self.currentSize:
            print('错误：下标已经超出数组范围！')
        if not self.heapList[i]:  # if None
            self.heapList[i] = key
            # print(self.heapList)
            while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
```

```
                temp = self.heapList[i]
                self.heapList[i] = self.heapList[self.parent(i)]
                self.heapList[self.parent(i)] = temp
                i = self.parent(i)
                # print(self.heapList)
            return
        elif key >= self.heapList[i]:
            self.heapList[i] = key
            while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
                temp = self.heapList[i]
                self.heapList[i] = self.heapList[self.parent(i)]
                self.heapList[self.parent(i)] = temp
            return
        elif key < self.heapList[i]:
            print('错误：输入的key值比原有的数值小！')
            print('当前heap的情况为',self.heapList)
            print('key=',key)
            print('position=',i)



def main():
    myHeap = d_ary_heap(3)
    for i in  range(1,31):
        myHeap.insert(i)
    print('1~30插入完成')
    myHeap.show_heap()
    myHeap.extra_max()
    myHeap.show_heap()
    myHeap.increase_key(10,28)
    print('将第10个元素的权重增加到28！')
    myHeap.show_heap()

if __name__ == '__main__':
    main()
```