

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	1
2	The height of a d-ary heap	1
3	The implementation of EXTRA-MAX function	1
4	The implementation of INSERT function	2
5	The implementation of INCREASE-KEY function	2
6	Constructing a 3-ary heap	3

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 satisfy:

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

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

```

1  def extra_max(self):
2      if self.currentSize < 1:
3          print('错误: heap已经空的! ')
4          return
5      max_item = self.heapList[1]
6      self.heapList[1] = self.heapList[self.currentSize]
7      self.currentSize -= 1
8      self.heapify(1)
9      print('弹出最大元素! 最大元素是: ',max_item)
10     return max_item
11
12     def heapify(self,i):
13         child_range = self.childs(i)
14         temp = None
15         largest = i
16         for k in child_range:
17             if k <=self.currentSize and self.heapList[k] > self.heapList[i]:
18                 largest = k
19         if largest != i:
20             temp = self.heapList[i]
21             self.heapList[i] = self.heapList[largest]
22             self.heapList[largest] = temp
23             self.heapify(largest)

```

The running time is determined by the process of heapifying $A[1]$. 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

```

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

```

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

5 The implementation of INCREASE-KEY function

Listing 3: The implementation of INCREASE-KEY

```

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

```

```

6         # print(self.heapList)
7         while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
8             temp = self.heapList[i]
9             self.heapList[i] = self.heapList[self.parent(i)]
10            self.heapList[self.parent(i)] = temp
11            i = self.parent(i)
12            # print(self.heapList)
13        return
14    elif key >= self.heapList[i]:
15        self.heapList[i] = key
16        while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
17            temp = self.heapList[i]
18            self.heapList[i] = self.heapList[self.parent(i)]
19            self.heapList[self.parent(i)] = temp
20        return
21    elif key < self.heapList[i]:
22        print('错误: 输入的key值比原有的数值小! ')
23        print('当前heap的情况为',self.heapList)
24        print('key=',key)
25        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:

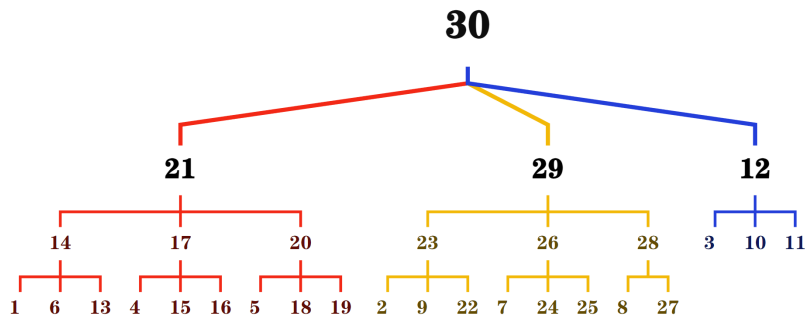


图 1: The heap after inserting a sequence ranging from 1 to 30

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

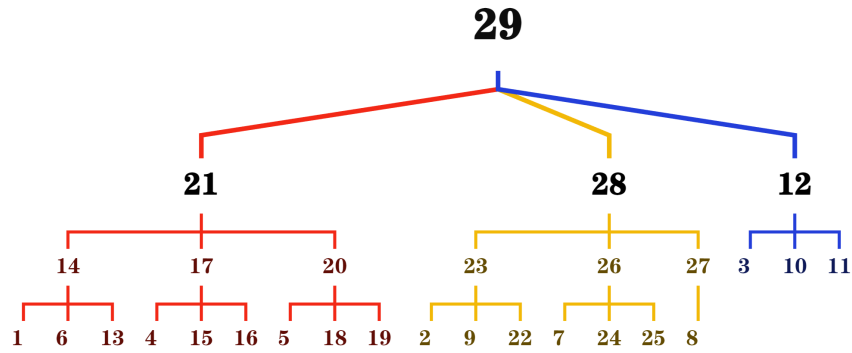


图 2: The heap after performing EXTRA-MAX

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

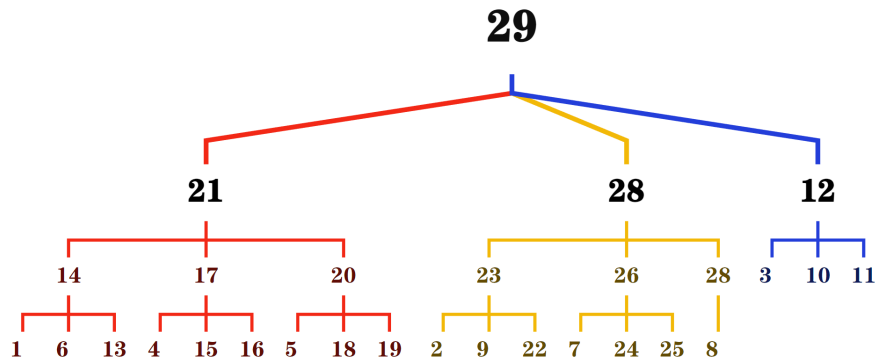


图 3: The heap after performing INCREASE-KEY(A, 10, 28)

Appendix: all of my python codes

Listing 4: All of my python codes

```

1 from math import *
2
3 class d_ary_heap:
4     def __init__(self,d):
5         self.heapList = [0]
6         self.d_factor = d
7         self.currentSize = 0
8
9     def show_heap(self):
10        d = self.d_factor
11        print('\n', str(d), '叉堆如下\n')
12        count = 1
13        for level in range(ceil(log(self.currentSize, d))):
14            for _ in range(d**level):
15                if count > self.currentSize:
16                    print('\n')

```

```

17         return
18     else:
19         print(self.heapList[count],end=' ')
20         count += 1
21     print('\n')
22
23
24
25
26 def childs(self,i):
27     temp = i * self.d_factor
28     return tuple( range(temp - (self.d_factor - 2), temp + 1 + 1))
29
30 def parent(self,i):
31     return (i + (self.d_factor-2))//self.d_factor
32
33 def extra_max(self):
34     if self.currentSize < 1:
35         print('错误: heap已经空的! ')
36         return
37     max_item = self.heapList[1]
38     self.heapList[1] = self.heapList[self.currentSize]
39     self.currentSize -= 1
40     self.heapify(1)
41     print('弹出最大元素! 最大元素是: ',max_item)
42     return max_item
43
44 def heapify(self,i):
45     child_range = self.childs(i)
46     temp = None
47     largest = i
48     for k in child_range:
49         if k <=self.currentSize and self.heapList[k] > self.heapList[i]:
50             largest = k
51     if largest != i:
52         temp = self.heapList[i]
53         self.heapList[i] = self.heapList[largest]
54         self.heapList[largest] = temp
55         self.heapify(largest)
56
57 def insert(self, key):
58     self.currentSize += 1
59     self.heapList.append(None)
60     self.increase_key(self.currentSize, key)
61
62 def increase_key(self, i, key):
63     if i > self.currentSize:
64         print('错误: 下标已经超出数组范围! ')
65     if not self.heapList[i]: # if None
66         self.heapList[i] = key
67         # print(self.heapList)
68         while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:

```

```

69         temp = self.heapList[i]
70         self.heapList[i] = self.heapList[self.parent(i)]
71         self.heapList[self.parent(i)] = temp
72         i = self.parent(i)
73         # print(self.heapList)
74         return
75     elif key >= self.heapList[i]:
76         self.heapList[i] = key
77         while i > 1 and self.heapList[self.parent(i)] < self.heapList[i]:
78             temp = self.heapList[i]
79             self.heapList[i] = self.heapList[self.parent(i)]
80             self.heapList[self.parent(i)] = temp
81         return
82     elif key < self.heapList[i]:
83         print('错误: 输入的key值比原有的数值小! ')
84         print('当前heap的情况为',self.heapList)
85         print('key=',key)
86         print('position=',i)
87
88
89
90 def main():
91     myHeap = d_ary_heap(3)
92     for i in range(1,31):
93         myHeap.insert(i)
94     print('1~30插入完成')
95     myHeap.show_heap()
96     myHeap.extra_max()
97     myHeap.show_heap()
98     myHeap.increase_key(10,28)
99     print('将第10个元素的权重增加到28! ')
100    myHeap.show_heap()
101
102 if __name__ == '__main__':
103     main()

```