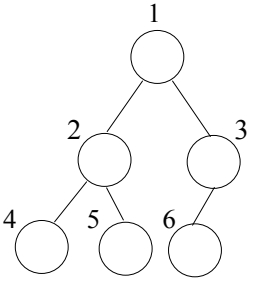
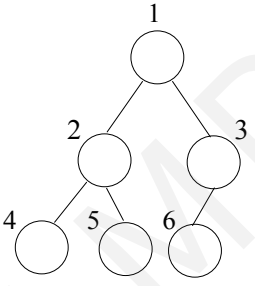
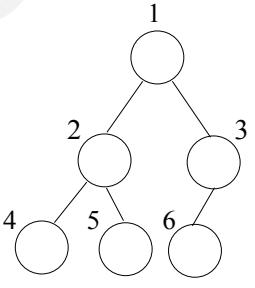
接下來，第 2 節點的子節點為第 4 和 5 節點，選最大值 40 (因為 40 大於 8)，且 40 又大於 23，故要交換，情形如下：



最後，第 1 節點的子節點為第 2 和 3 節點，其值分別為 40，20，選最大的 40 與父節點(第 1 節點)的值 15 相比，40 大於 15，所以要交換，結果如下：

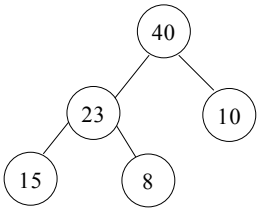


記得對調後要繼續往下與最大的子節點相比，看看是否還要對調。當 15 和 40 交換後，15 需要再和其最大子節點的鍵值(23)相比，由於 23 大於 15，故需要再交換，最後的結果如下圖所示:



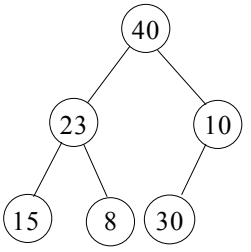
7.1.1 Heap 的加入

假設有一棵 Heap 如下：

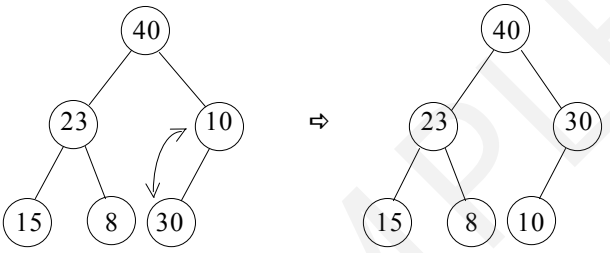


我們以加入 30 及 50 來說明其調整的過程。

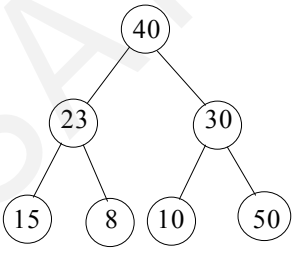
首先，依照完整二元樹的特性將 30 加進來，如下圖所示：



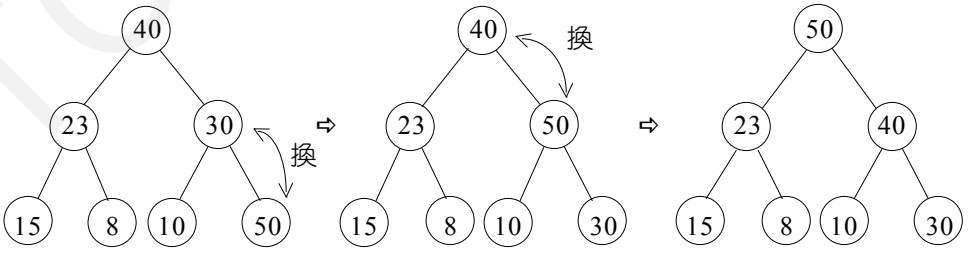
由於 30 大於 10，所以要加以調整，如下圖所示：



接著加入 50

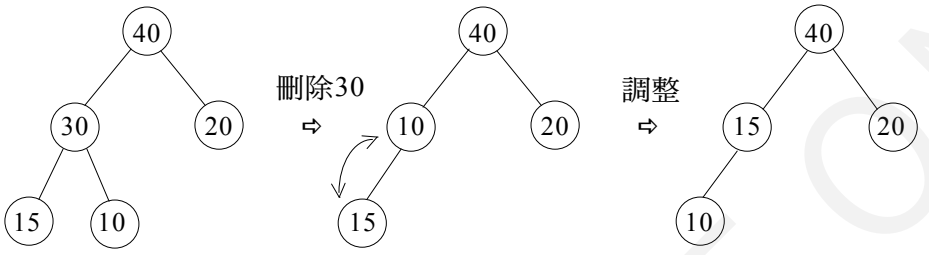


因為不符合 Heap 的定義，所以需加以調整之，如下圖所示:

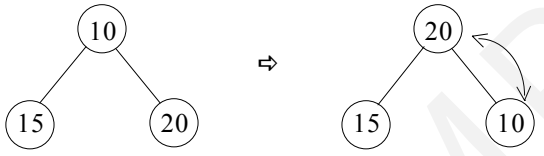


7.1.2 Heap 的刪除

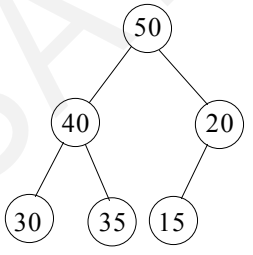
Heap 的刪除，首先以完整二元樹的最後一節點取代被刪除的節點，然後判斷是否為一棵 Heap，若不是，則要加以調整。請看以下的範例。有一棵 Heap 如下，首先刪除 30，此時將以 10 來取代，並加以調整之。



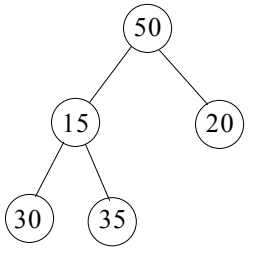
再刪除 40，此時以 10 取代之，且將子節點中最大者(20)和父節點(10)互相對調。



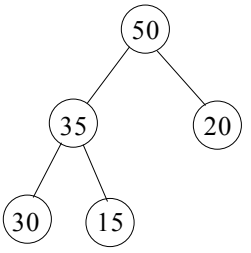
再看一範例，若要刪除下一棵 Heap 的 40



首先，以 15 取代 40 (因為 15 在完整二元樹中是最後一個節點)



接著將 15 和其所屬的最大子節點(35)比較，由於 35 大於 15，故將 15 和 35 交換

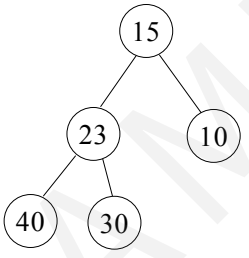


之後，再與 35 的父節點(50)比較，因為 35 小於 50，所以不需交換。

有關 Heap 的程式實作，請參閱 7-4 節。

練習題

1. 請將下列的二元樹調整為一棵 Heap



2. 將下列的資料建立成一棵 Heap。

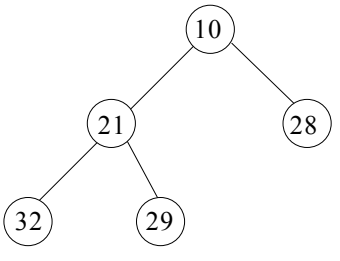
   20，30，10，50，60，40，45，5

   [提示：先將它建立成一棵完整二元樹，之後再依據 Heap 的特性調整之。]

3. 承 2，將已建立的 Heap，分別刪除 30 和 60。

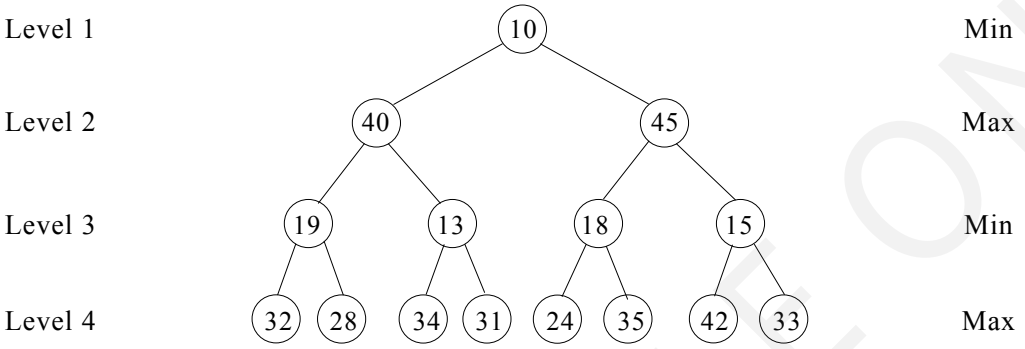
7.2 Min-Max heap

以上所介紹的 Heap 稱之為 Max heap。在 Max heap 中，父節點的鍵值大於子節點鍵值; 反之，若父節點的鍵值小於子節點的鍵值，則稱它為 Min heap，如下圖所示。



由於其加入與刪除的方法與 Max heap 相似，在此就不再贅述了。除了上述的 Max heap 和 Min heap 外、還有 Min-Max heap 和 Deap，我們將分別在此節和 7.3 節加以說明。

Min-Max heap 包含了 Min heap 與 Max heap 的特徵，如下圖所示：

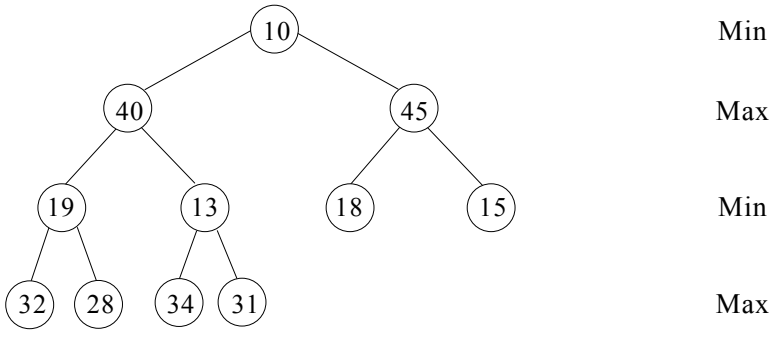


我們就直接以上圖來說明 Min-Max heap 的定義。

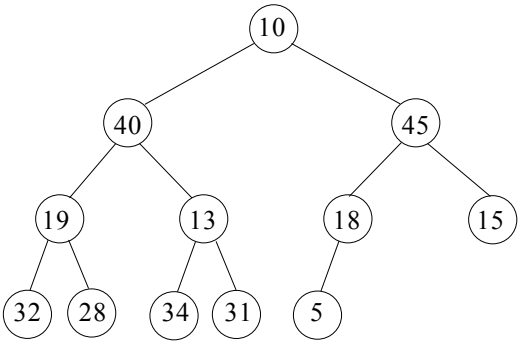
1. Min-Max heap 是以一層 Min heap，一層 Max heap 交互構成的，如 Level 1 中各節點的鍵值一律小於它的子節點(10 小於 40、45)，Level 2 中各節點的鍵值一律大於它的子節點(40 大於 19、13；45 大於 18、15)，而 Level 3 的節點鍵值又小於它的子節點(19 小於 32、28；13 小於 34、31；18 小於 24、35；15 小於 42、33)。
2. 樹中為 Min heap 的部份，仍需符合 Min heap 的特性，如上圖中 Level 1 的節點鍵值小於 Level 為 3 的子樹(10 小於 19、13、18、15)。
3. 樹中為 Max heap 的部份，仍需符合 Max heap 的特性，如上圖中 Level 2 的節點鍵值大於 Level 為 4 之子樹(40 大於 32、28、34、31；45 大於 24、35、42、33)。

7.2.1 Min-Max heap 的加入

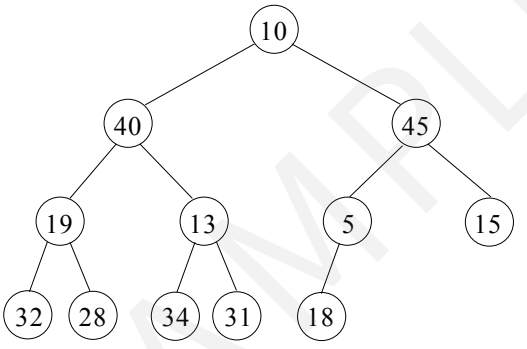
Min-Max heap 的加入與 Max heap 的原理差不多，加入後要調整至符合上述 Min-Max heap 的定義。假設有一棵 Min-Max heap 如下：



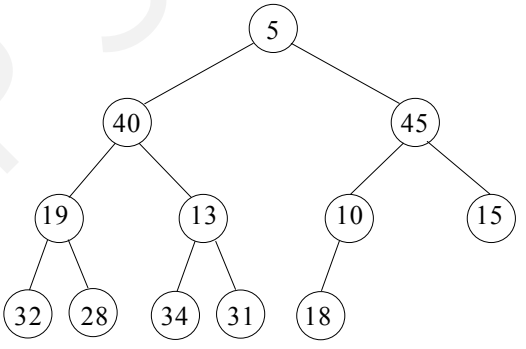
加入 5 的步驟如下：



加入後，由於 18 < 5，不符合第(1)項定義，需將 5 與 18 交換

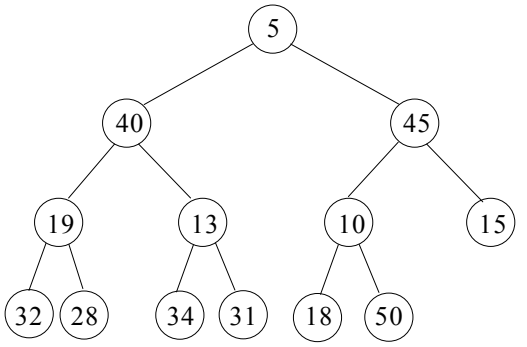


交換後，由於 10 < 5，不符合第(2)項定義，需將 5 與 10 對調

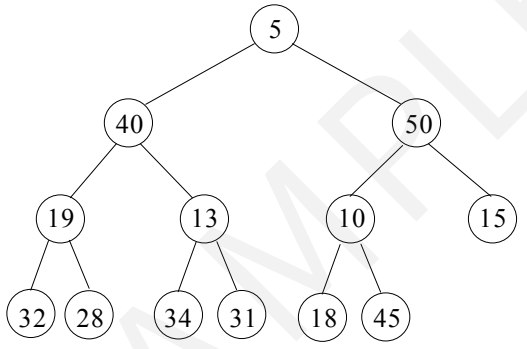


上圖已符合 Min-Max heap 的定義，所以不需再調整。

承上，再加入 50，其步驟如下：



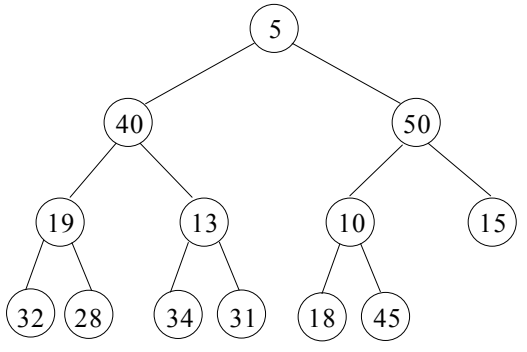
加入後 45 < 50，不符合第(3)項定義，需將 45 與 50 交換



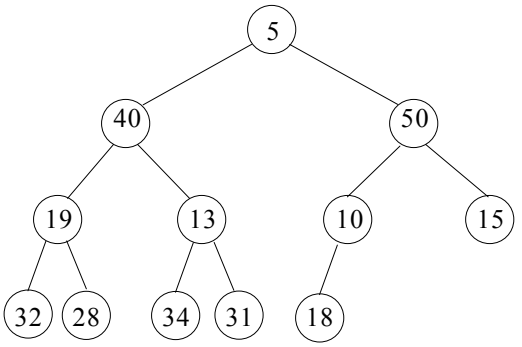
上圖已符合 Min-Max heap 的定義，因此不需再調整。

7.2.2 Min-Max heap 的刪除

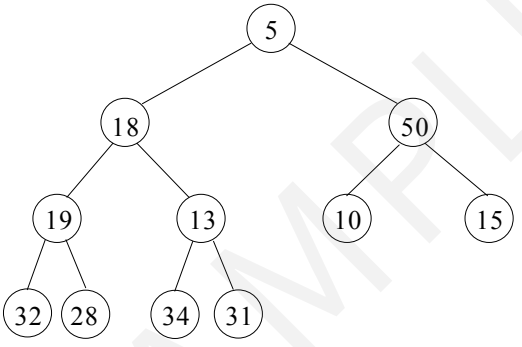
若刪除 Min-Max heap 的最後一個節點，則直接刪除即可；否則，先以樹中最後一個編號的節點，取代被刪除節點，之後再作調整動作。假設已存在一棵 Min-Max heap 如下：



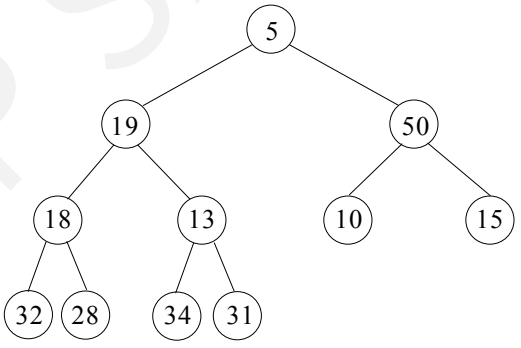
刪除 45，則直接刪除。



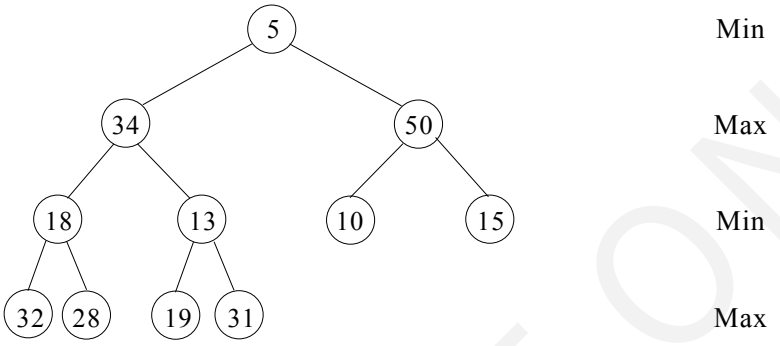
再刪除 40，則需以最後一個節點的鍵值 18 來取代 40



交換後 18 < 19，不符合第(1)項定義，需將 18 與 19 交換



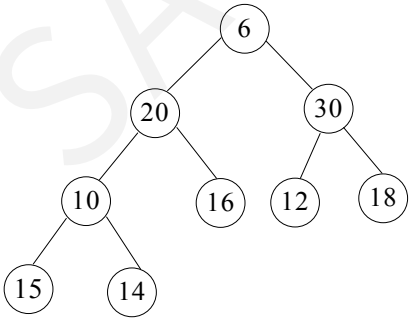
交換後，由於 19 小於 32、28、34、31，不符合第(3)項定義，需將 19 與最大的鍵值 34 交換



此時已符合 Min-Max heap 的定義，所以不必再做調整。

練習題

1. 利用 7.1 節練習題第 2 題的資料，建立成一棵 Min heap。
2. 有一棵 Min-Max heap 如下：
   (a) 請依序加入 17，8 和 2，並畫出其所對應的 Min-Max heap。
   (b) 承(a)，依序刪除 20 和 10。

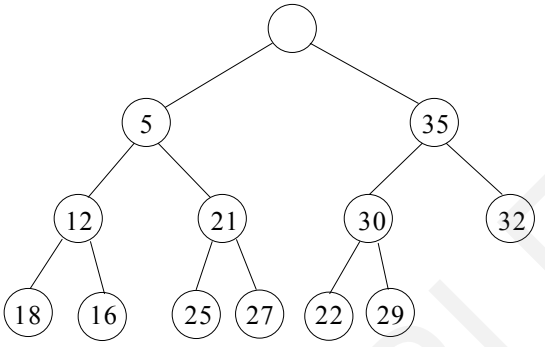


7.3 Deap

Deap 同時具備 Max heap 與 Min heap 的特徵，其定義如下：

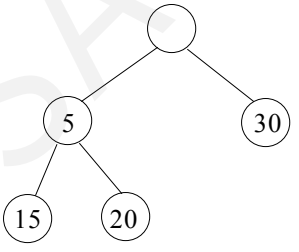
1. Deap 的樹根不儲存任何資料，為一空節點。
2. 樹根的左子樹是一棵 Min heap；右子樹則是一棵 Max heap。

3. Min heap 與 Max heap 中存在一對應的關係，假設左子樹中有一節點為 i，則在右子樹中必存在一節點 j 與之對應，此時 i 必需小於等於 j。如下圖中的 5 與 35 對應；12 與 30 對應，18 與 22 對應，16 與 29 對應，21 與 32 對應。那麼 25 與右子樹中的哪一個節點對應呢？當某一節點在右子樹中找不到對應節點時，該節點會與對應的右子樹之父節點相對應，所以 25 會與右子樹中鍵值為 32 的節點相對應，25 小於 32。

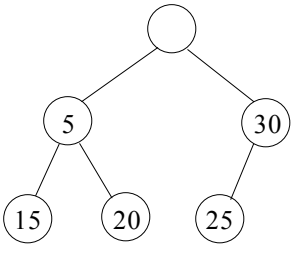


7.3.1 Deap 的加入

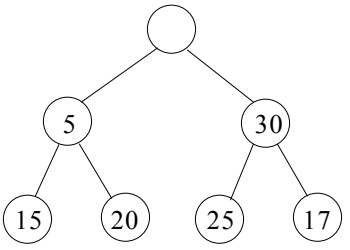
Deap 的加入動作與 Heap 相同，將新鍵值加入於整棵樹的最後，再調整至符合 Heap 的定義，我們以一範例來說明 Deap 的加入。假設已存在一 Deap 如下：



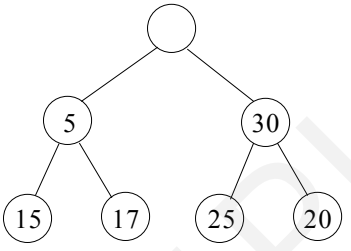
若加入 25，加入後右子樹仍為一棵 Max heap，且左子樹對應節點 15 小於等於右子樹節點 25，符合 Deap 的定義，如下圖所示：



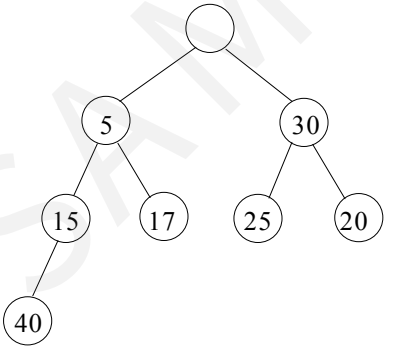
加入 17。加入後的圖形如下所示：



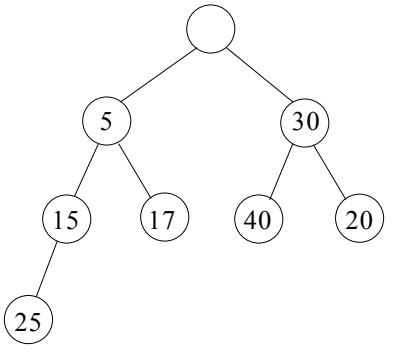
此時右子樹仍為 Max heap，但 17 小於其左子樹的對應節點 20，故將 17 與 20 交換



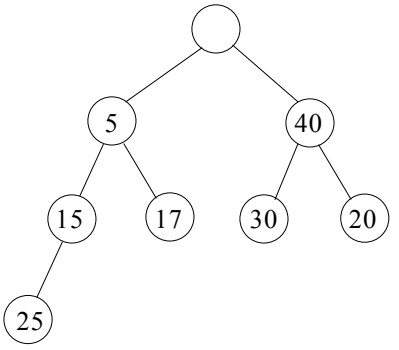
加入 40，如下圖所示：



加入後，左子樹雖為 Min heap，但 40 大於其所對應節點 25 (與節點 40 所對應右子樹節點的父節點)，不符合 Deap 定義，故需將 40 與 25 交換，如下圖所示：

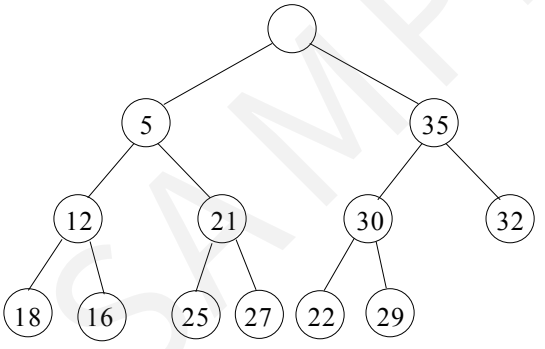


交換後，右子樹不是一棵 Max heap，需將 40 與 30 對調，如下圖所示：

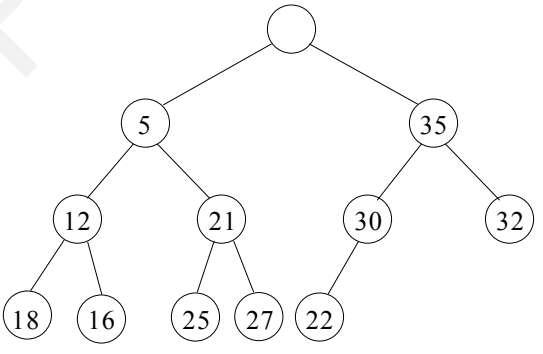


7.3.2 Deap 的刪除

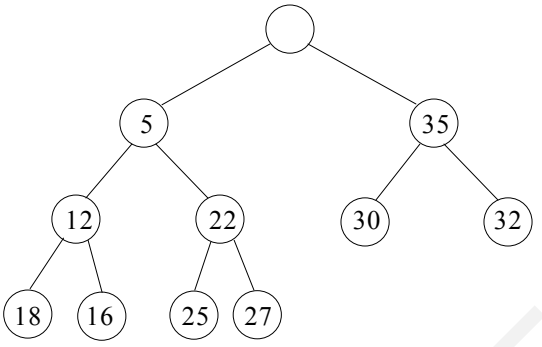
Deap 的刪除動作與 Heap 一樣，當遇到刪除節點非最後一個節點時，要以最後一個節點的鍵值取代刪除節點，並加以調整之，直到符合 Deap 的定義為止。假設有一棵 Deap 如下：



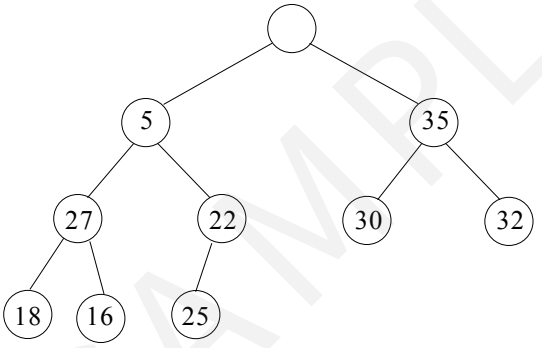
若刪除 29，則直接刪除之，如下圖所示：



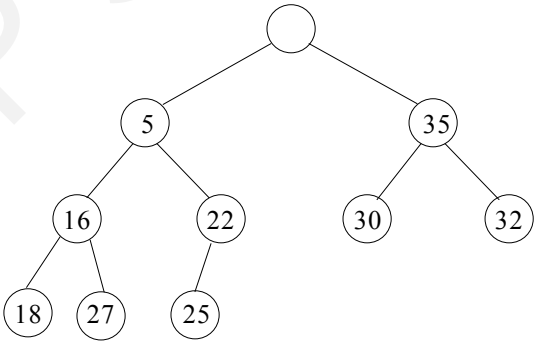
若刪除 21，則以最後一個節點 22 取代之。檢查左子樹仍為一棵 Min heap，且節點鍵值 22 小於其對應節點 32，所以不需要調整。結果如下圖所示：



再刪除 12，並以最後一個節點 27 取代



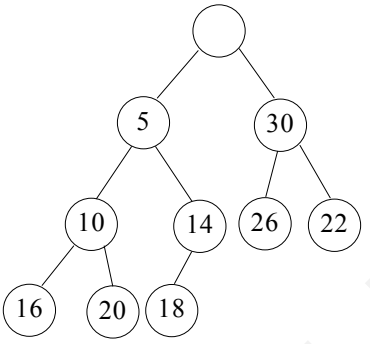
由於左子樹不符合 Min heap 的定義，故需將 27 與其子節點中最小者(16)交換，如下圖所示:



由於 16 小於其對應的節點 30，所以不需再調整。

練習題

1. 有一棵 Deap 如下：



(a) 請依序畫出加入 2 和 50 所對應的 Deap。

(b) 承(a)，畫出刪除 50 後的 Deap。

7.4 程式實作

(一) 利用 Heap 處理會員進出資料

Python 程式語言實作》利用 Heap 處理會員進出資料

```
# 利用Heap 處理會員推出資料——新增、刪除、輸出
# File Name: HeapTree.py
# Version 3.0, March 13th, 2017

import sys

MAX = 100
heap_tree = [0] * MAX # Heap 串列
last_index = 0 # 最後一筆資料的 index

# 新增函數
def insert_f():
    global MAX
    global last_index

    if last_index >= MAX-1: # 資料數超過上限，顯示錯誤訊息
```

```
17          print('\n   Login members are more than %d !!' % MAX - 1)
18          print('   Please wait for a Minute ...')
19
20      id_temp = int(input('\n   Please enter login ID number: '))
21      create(id_temp) # 建立Heap
22      print('     Login successfully!!')
23
24  # 刪除函數
25  def delete_f():
26      global last_index
27
28      id_temp = 0
29      del_index = 0
30
31      if last_index < 1: # 無資料存在，顯示錯誤訊息
32          print('\n   No member to logout!!')
33          print('   Please check again!!')
34      else:
35          id_temp = int(input('\n   Please enter logout ID number: '))
36          del_index = search(id_temp)
37          if del_index == 0: # 沒找到資料，顯示錯誤訊息
38              print('   ID number not found!!')
39          else:
40              removes(del_index) # 刪除資料，並調整Heap
41              print('   ID number ', id_temp, ' logout!!')
42
43  # 輸出函數
44  def display_f():
45      global last_index
46      option = ''
47
48      if last_index < 1: # 無資料存在，顯示錯誤訊息
49          print('\n   No member to show!!\n')
50      else:
51          print()
52          print('********************')
53          print('   <1> increase') # 選擇第一項為由小到大排列
54          print('   <2> decrease') # 選擇第二項為由大到小排列
55          print('********************')
56          while True:
57              try:
58                  option = input('\n   Please enter your option: ')
59              except ValueError:
60                  print()
```

```
                print('Not a correctly number.')
                print('Try again\n')
            if (option == '1' or option == '2'):
                break
        show(option)

# 建立資料於Heap，ID_TEMP 為新增資料
def create(id_temp):
    global last_index
    global heap_tree
    last_index += 1
    heap_tree[last_index] = id_temp # 將資料新增於最後
    adjust_u(heap_tree, last_index) # 調整新增資料

# 從Heap 中刪除資料，INDEX_TEMP 為欲刪除資料之 INDEX
def removes(index_temp):
    global last_index
    global heap_tree

    # 以最後一筆資料代替刪除資料
    heap_tree[index_temp] = heap_tree[last_index]
    heap_tree[last_index] = 0
    last_index -= 1
    if last_index > 1: # 當資料筆數大於 1 筆，則做調整
        # 當替代資料大於其 PARENT NODE，則往上調整
        if heap_tree[index_temp] > heap_tree[index_temp//2] and index_temp > 1:
            adjust_u(heap_tree, index_temp)
        else: # 替代資料小於其 CHILDREN NODE，則往下調整
            adjust_d(heap_tree, index_temp, last_index-1)

# 印出資料於螢幕
def show(op):
    global last_index
    global heap_tree
    heap_temp = []
    tChar = ''

    # 將 Heap 資料複製到另一個串列作排序工作
    heap_temp = [i for i in heap_tree]
    # 將串列調整為由小到大排列
    c_index = last_index - 1
    while c_index > 0:
        exchange(heap_temp, 1, c_index+1)
        adjust_d(heap_temp, 1, c_index)
```