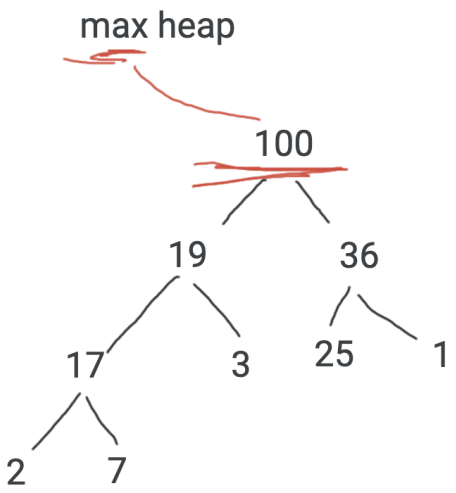
Phill-DS-0220 0306

Heap

- tree, binary tree 不是 binary search tree → 上下有大小 , 左右不分大小
- 四種 heap → max heap , min heap, min-max heap, deap
- 應用
  - heap sort → efficient nlogn
  - priority queue 的陽春版

Max heap

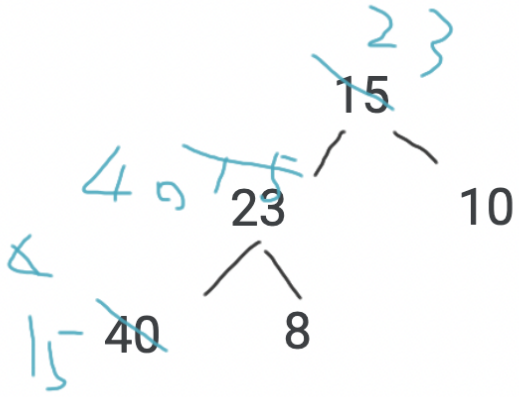


- 大小
- 排列方式 保證 最大值永遠在根節點
  - priority queue
  - O(nlogn)
  - sorting

建立 heap

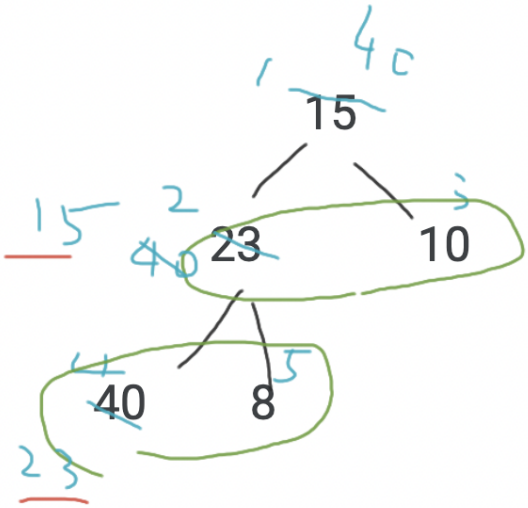
由上到下(調整)

- 鐵三角比較： 上下比兩次 or 左右比再上比
- 逐層比較由上往下
- 比完再向上比一次 以免造成層與層之間大小不對的狀況



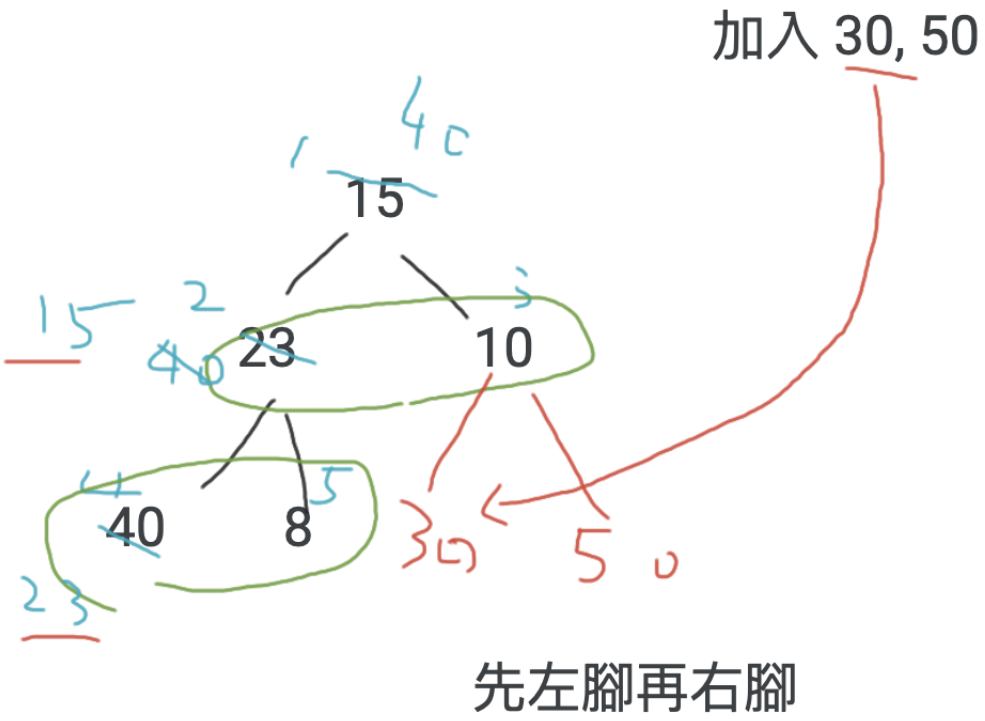
由下往上

- 對節點做編號 以大號碼開始處理
- 鐵三角同層比較 由下往上進行
- 比完以後還要向下比一次



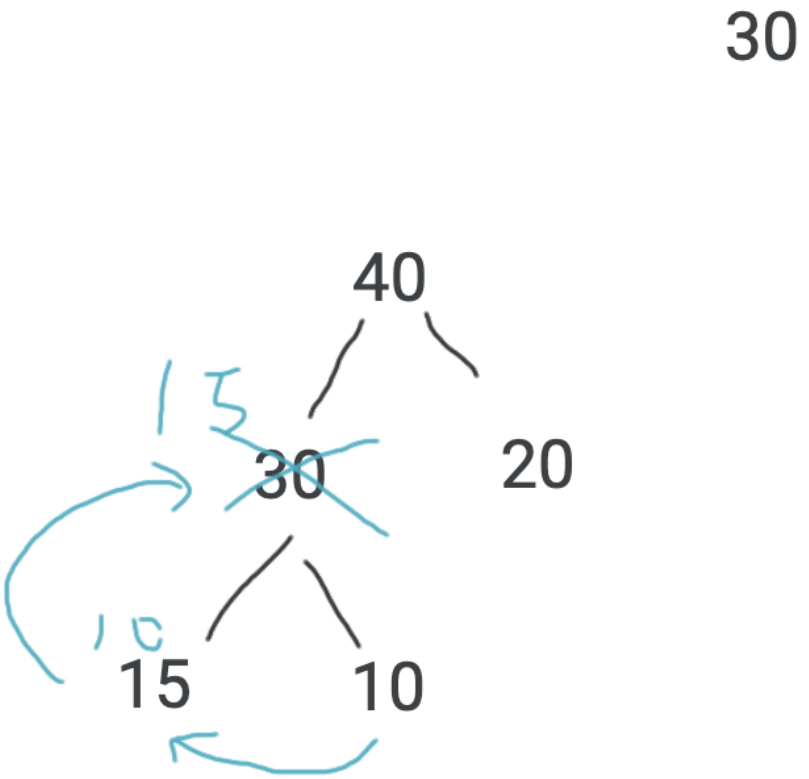
加入元素

- 要加入的元素加入最下方
- 藉由調整的機制還原成 heap 的原則
- note: 實作上 希望先加左 再加右



刪除元素

- 刪除節點
- 先提左腳
- 轉右腳到左腳(維持先左後右的原則)



實作

```
#include <iostream>
using namespace std;

class MaxHeap{
  private:
    int* heapArray; //指標開頭將產生實體
    int capacity;
    int currentSize;

    void shiftUp(int index){
      int temp = heapArray[index]; //把現在的值放進魁儡變數
      while(index>0){ //因為到root 要停
        int parentIndex = (index-1) /2;
        if(heapArray[parentIndex] < temp){ //要換
```

```
            heapArray[index] = heapArray[parentIndex]; //換
            index = parentIndex;
          }
          else{
            break;
          }
        }
        heapArray[index] = temp; //我的最高點
      }

      void shiftDown(int index){
        int temp = heapArray[index];
        int childIndex = 2*index +1;
        while(childIndex < currentSize){
          if(temp < heapArray[childIndex]){
            heapArray[index] = heapArray[childIndex];
            index = childIndex;
            childIndex = 2*index +1;
          }
          else{
            break;
          }
        }
        heapArray[index] = temp; //我的最低點
      }

    public:
      MaxHeap(int cap): capacity(cap), currentSize(0){
        heapArray = new int[capacity];
      }

      ~MaxHeap(){
        delete[] heapArray; //c++11 連續空間歸還語法
      }

      void insert(int val){
```

```
      if(currentSize == capacity){
        cout << "滿了" << endl;
        return;
      }
      heapArray[currentSize] = val; //插入最後一個 先左再右的原則
      shiftUp(currentSize); //向上調整
      currentSize++; //多了一個
    }

    int removeMax(){
      int maxItem = heapArray[0];
      heapArray[0] = heapArray[currentSize-1]; //把最下面的提上來至
      currentSize--;
      shiftDown(0);
      return maxItem;
    }

        bool isEmpty(){
            return currentSize==0;
        }
};
```

Heap 的應用

- 使用上面的 class
- Heap Sort (由大到小)

```
int numbers[] = {8,5,2,9,10,6,3};
```

```
#include <iostream>
using namespace std;

class MaxHeap{
  private:
```

```
    int* heapArray; //指標開頭將產生實體
    int capacity;
    int currentSize;

    void shiftUp(int index){
      int temp = heapArray[index]; //把現在的值放進魁儡變數
      while(index>0){ //因為到root 要停
        int parentIndex = (index-1) /2;
        if(heapArray[parentIndex] < temp){ //要換
          heapArray[index] = heapArray[parentIndex]; //換
          index = parentIndex;
        }
        else{
          break;
        }
      }
      heapArray[index] = temp; //我的最高點
    }

    void shiftDown(int index){
      int temp = heapArray[index];
      int childIndex = 2*index +1;
      while(childIndex < currentSize){
        if(temp < heapArray[childIndex]){
          heapArray[index] = heapArray[childIndex];
          index = childIndex;
          childIndex = 2*index +1;
        }
        else{
          break;
        }
      }
      heapArray[index] = temp; //我的最低點
    }

  public:
```

```
    MaxHeap(int cap): capacity(cap), currentSize(0){
      heapArray = new int[capacity];
    }

    ~MaxHeap(){
      delete[] heapArray; //c++11 連續空間歸還語法
    }

    void insert(int val){
      if(currentSize == capacity){
        cout << "滿了" << endl;
        return;
      }
      heapArray[currentSize] = val; //插入最後一個 先左再右的原則
      shiftUp(currentSize); //向上調整
      currentSize++; //多了一個
    }

    int removeMax(){
      int maxItem = heapArray[0];
      heapArray[0] = heapArray[currentSize-1]; //把最下面的提上來至
      currentSize--;
      shiftDown(0);
      return maxItem;
    }

    bool isEmpty(){
            return currentSize==0;
        }
};

int main() {
    int numbers[] = {8,5,2,9,10,6,3};
    int length = sizeof(numbers)/sizeof(numbers[0]);

    MaxHeap heap(length);
```

```
    for(int i=0; i<length;i++)
      heap.insert(numbers[i]); //每次插入後會自動調整 heapify

    int j=0;
    while(!heap.isEmpty()){ //提出到空就停止
      numbers[j++]=heap.removeMax();
    }

    for(int z=0; z<length; z++)
      cout << numbers[z]<< " ";

    return 0;
}
```

練習

Priority Queue

- item 會有一個優先值 20, 5, 15, 40, 10
- class PriorityQueue → 使用 class MaxHeap
  - enqueue : 照順序 enqueue
  - dequeue : 依照優先權最高的取得

```
#include <iostream>
using namespace std;

class MaxHeap{
  private:
    int* heapArray;
    int capacity;
    int currentSize;

    void shiftUp(int index){
      int temp = heapArray[index];
```

```
        while(index > 0){
          int parentIndex = (index-1) / 2; //*** -- 父節點的正確索引
          if(heapArray[parentIndex] < temp){
            heapArray[index] = heapArray[parentIndex];
            index = parentIndex;
          }
          else{
            break;
          }
        }
        heapArray[index] = temp;
      }

      void shiftDown(int index){
        int temp = heapArray[index];
        while(index < currentSize){
          int leftChildIndex = 2 * index + 1;
          int rightChildIndex = 2 * index + 2;
          int largest = index;

          if(leftChildIndex < currentSize && heapArray[leftChildIn
            largest = leftChildIndex;
          }

          if(rightChildIndex < currentSize && heapArray[rightChild
            largest = rightChildIndex;
          }

          if(largest != index){
            swap(heapArray[index], heapArray[largest]); //*** -- 1
            index = largest;
          }
          else{
            break;
          }
        }
```

```
    }

  public:
    MaxHeap(int cap): capacity(cap), currentSize(0){
      heapArray = new int[capacity];
    }

    ~MaxHeap(){
      delete[] heapArray;
    }

    void insert(int val){
      if(currentSize == capacity){
        cout << "Heap is full" << endl; //*** -- heap已滿的提示信息
        return;
      }
      heapArray[currentSize] = val;
      shiftUp(currentSize);
      currentSize++;
    }

    int removeMax(){
      if(isEmpty()){ //*** -- 檢查是否為空，以避免在空堆上進行操作
        return -1; // Error code
      }
      int maxItem = heapArray[0];
      heapArray[0] = heapArray[currentSize-1];
      currentSize--;
      shiftDown(0);
      return maxItem;
    }

    bool isEmpty(){
            return currentSize == 0;
        }
};
```

```
class PriorityQueue{
  private:
    MaxHeap maxHeap;

  public:
    PriorityQueue(int capacity): maxHeap(capacity){}

    void enqueue(int value){
      //*** -- 刪除了不必要的空條件判斷，因為MaxHeap已經有滿的檢查
      maxHeap.insert(value);
      cout << "Priority "<< value << " has been enqueued." << en
    }

    int dequeue(){
      int value = maxHeap.removeMax(); //優先權最高
      if(value == -1){
          cout << "Queue is empty." << endl;
      }
      return value;
    }

    bool isEmpty(){
      return maxHeap.isEmpty();
    }
};

int main() {
    PriorityQueue q(10);

    q.enqueue(20);
    q.enqueue(5);
    q.enqueue(15);
    q.enqueue(40);
    q.enqueue(10);
```

```
    while(!q.isEmpty()){
        int dequeuedValue = q.dequeue();
        if(dequeuedValue != -1){ //*** -- 檢查dequeue返回值，避免錯
            cout << "Dequeued: " << dequeuedValue << endl;
        }
    }
    return 0;
}
```