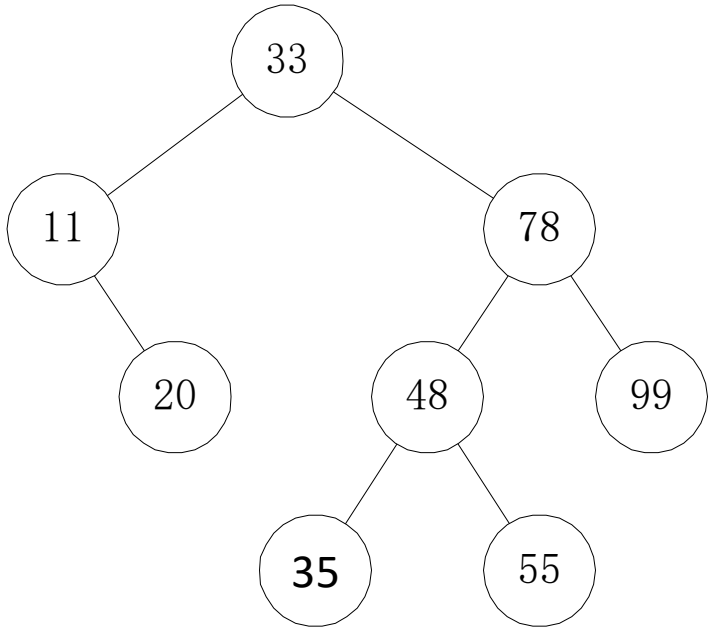
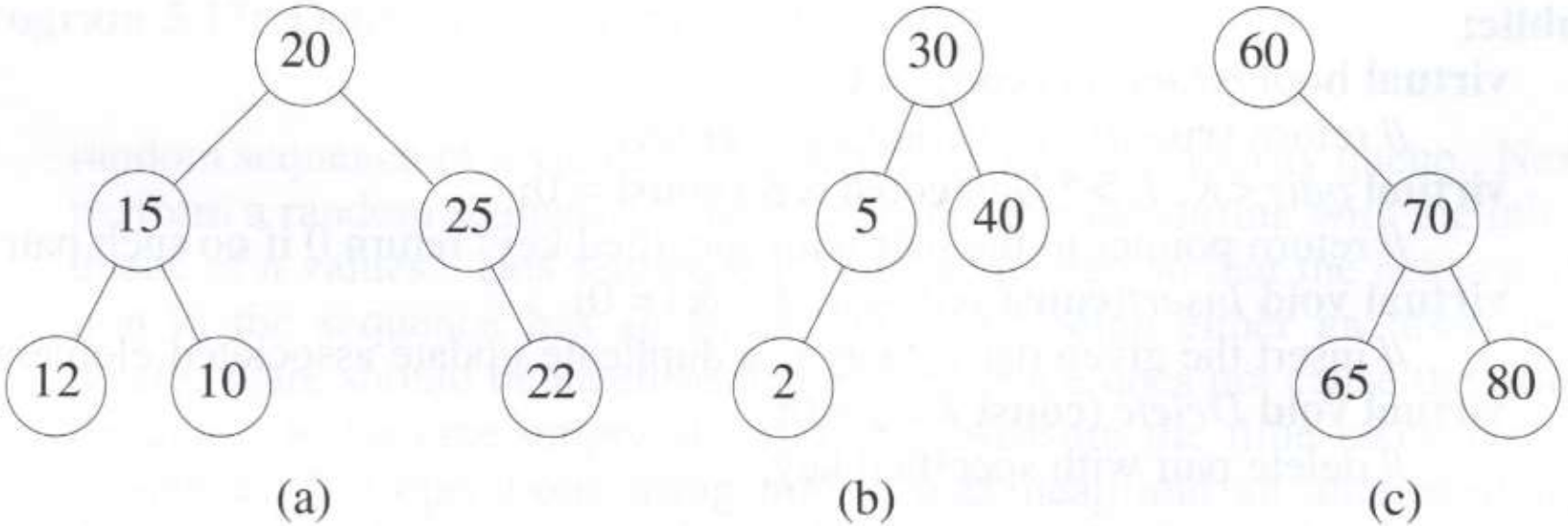
Chapter 5 樹狀結構(2)

二元搜尋樹(Binary Search Tree)

- 二元搜尋樹是一個二元樹，它有可能是空的，而如不是空，就必須滿足下列性質：
  1. 每一個節點上均存放一個鍵值資料，且沒有兩個節點的鍵值資料相同
  2. 樹根之鍵值大於左子樹的所有鍵值
  3. 樹根之鍵值小於右子樹的所有鍵值
  4. 左右子樹也是二元搜尋樹。

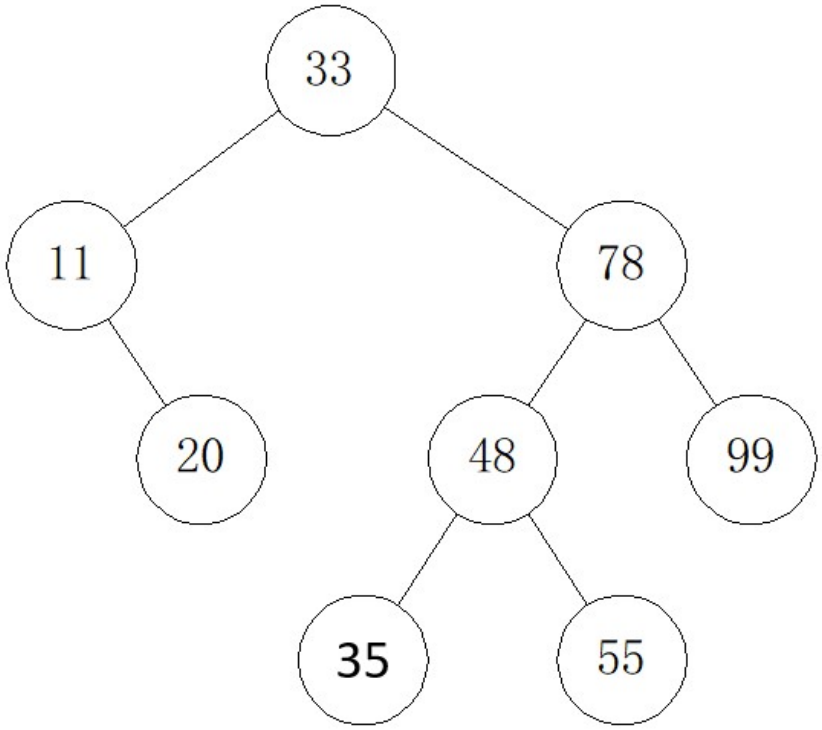


練習
哪些是二元搜尋樹



二元搜尋樹的新增(1)

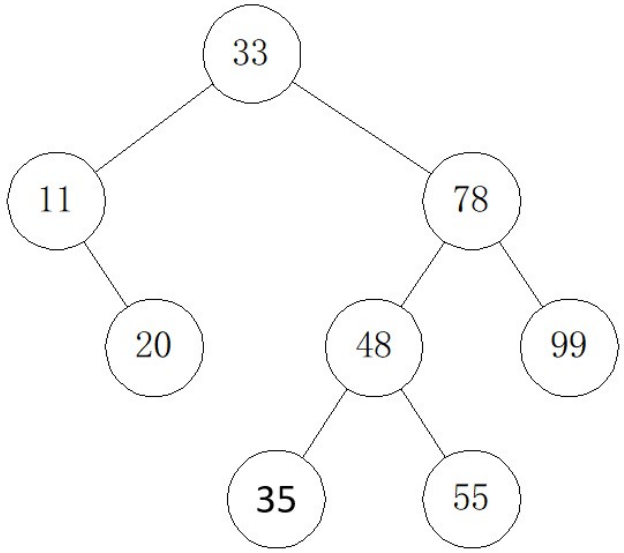
- 設一組等待輸入的資料{33, 78, 11, 48, 20, 55, 35, 99}



二元搜尋樹的新增(2)

```
public boolean insert(TreeNode node){
    TreeNode curNode = root;
    TreeNode parentNode = null;
    while(curNode!=null) {
        parentNode = curNode;
        if(node.key == curNode.key)
                return false;
        if(node.key<curNode.key)
                curNode=curNode.leftChild;
        else
                curNode = curNode.rightChild;
    }
    curNode = node;
    if(root == null)
        root = curNode;
    else if (curNode.key<parentNode.key)
        parentNode.leftChild = curNode;
    else
        parentNode.rightChild = curNode;
    return true;
}
```

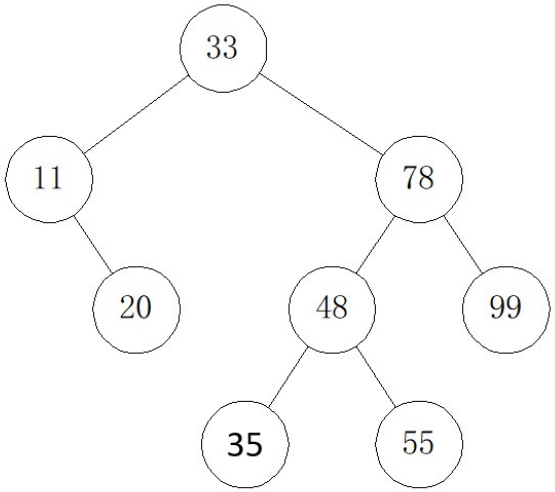
{33, 78, 11, 48, 20, 55, 35, 99}



作業05-06

- 請完成二元搜尋樹的新增節點方法

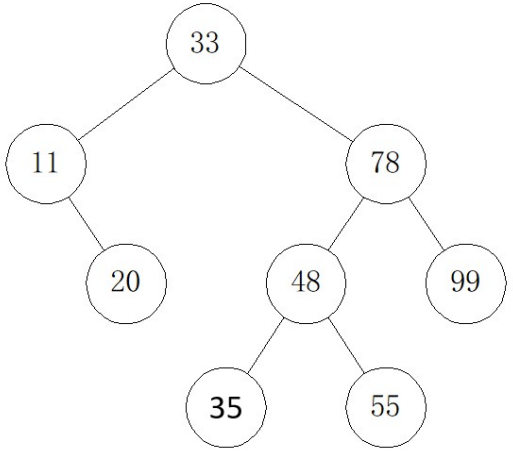
二元搜尋樹的搜尋(遞迴版)
找某一特定鍵值key的節點




```
public TreeNode findKeyTreeNode(int key) {
        return findKeyTreeNode(root,key);
}
public TreeNode findKeyTreeNodeRecursive(TreeNode curNode,int key) {
    if(curNode == null)
        return null;
    if(key == curNode.key)
        return curNode;
    if(key<curNode.key)
        return findKeyTreeNodeRecursive(curNode.leftChild, key);
    else
        return findKeyTreeNodeRecursive(curNode.rightChild, key);
}
```

二元搜尋樹的搜尋(非遞迴版)

```
public TreeNode findKeyTreeNode(int key) {
        return findKeyTreeNode(root,key);
}

public TreeNode findKeyTreeNode(TreeNode curNode,int key) {
    while(curNode!=null) {
        if(curNode.key == key)
                return curNode;
        else if(key<curNode.key) {
                curNode = curNode.leftChild;
        }
        else
        {
                curNode = curNode.rightChild;
        }
    }
    return null;
}
```



作業05-07

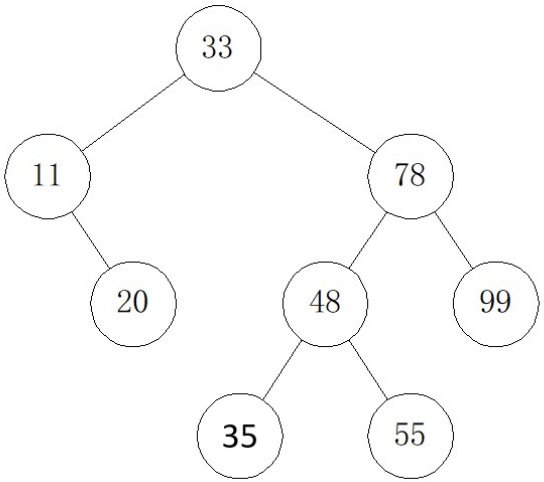
- 請完成二元搜尋樹的搜尋遞迴版

作業05-08

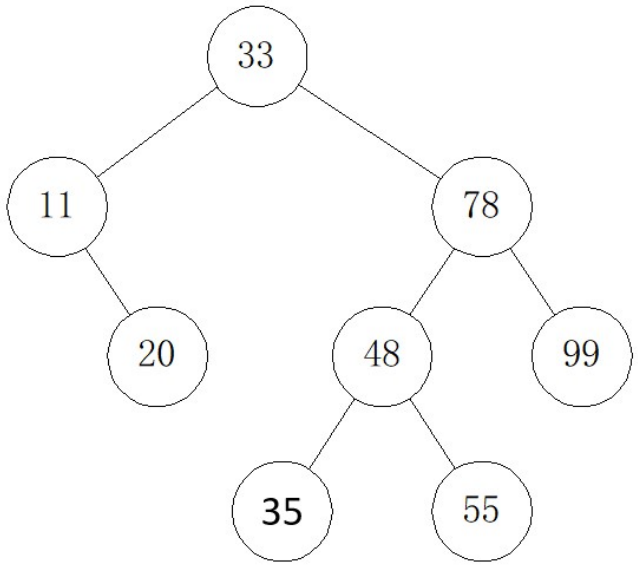
- 請完成二元搜尋樹的搜尋非遞迴版

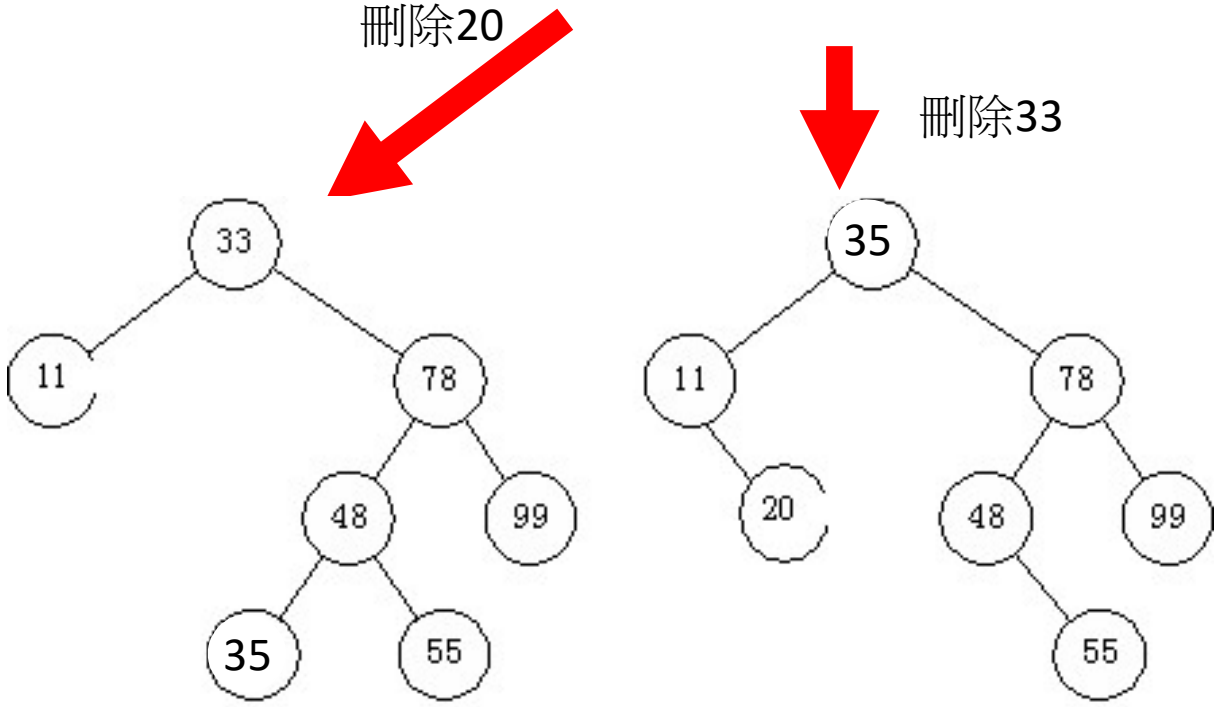
二元搜尋樹刪除(1)

- 二元搜尋樹的刪除
  1. 刪除的節點為樹葉節點則直接刪除。
  2. 刪除的節點有一個子節點，則將原本指向刪除節點的link指向此子節點
  3. 刪除的節點有兩個子節點，則將刪除的節點以左子樹最大的節點或右子樹最小的節點取代。
  4. 取代的節點可能沒有子節點(使用步驟1，直接刪除取代的節點)，可能有一個子節點(使用步驟2，將原本指向取代節點的link指向此子節點)



二元搜尋樹刪除(2)


33
11
78
20
48
99
35
55

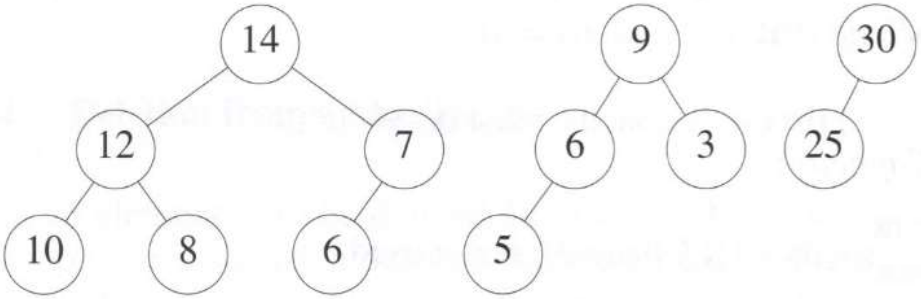

刪除20
刪除33
33
11
78
48
99
35
55
35
11
78
20
48
99
55

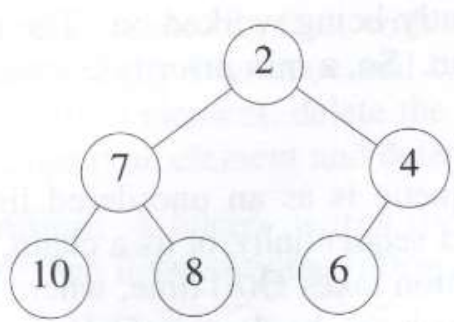
進階作業05-01

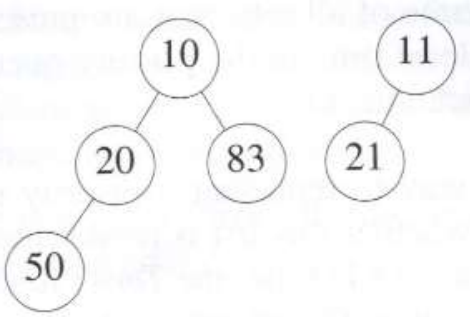
- 請完成二元搜尋樹刪除的方法

最大(小)堆積(Max(Min) Heap)

- 最大(小)樹 (max(min) tree)
  - 最大(小)樹 (max(min) tree)是指一個樹它每一個節點的鍵值大於(小於)其子節點的關鍵值。
- 最大(小)堆積
  - 是一個最大(小)樹，且符合完整二元樹的定義

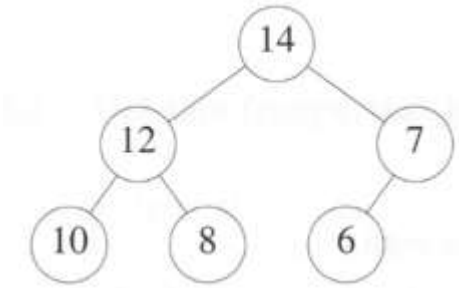






Max Heap ADT

- 最大(小)堆積是一個完整二元樹，所以以陣列實作




```
public class MaxHeap {
    int[] heap;
    int heapSize;
    //current size of max heap
    int capacity;
    //Maximum allowable size of heap
    MaxHeap(int theCapacity);
    //Create an empty heap that can hold a maximum of
    capacity elements
    public boolean isFull();
    public boolean isEmpty();
    public void push(int newValue);
    //insert a item into the heap.
    public void pop();
    //remove the largest item from the heap
}
```

Max Heap ADT implementation (1)
建構子

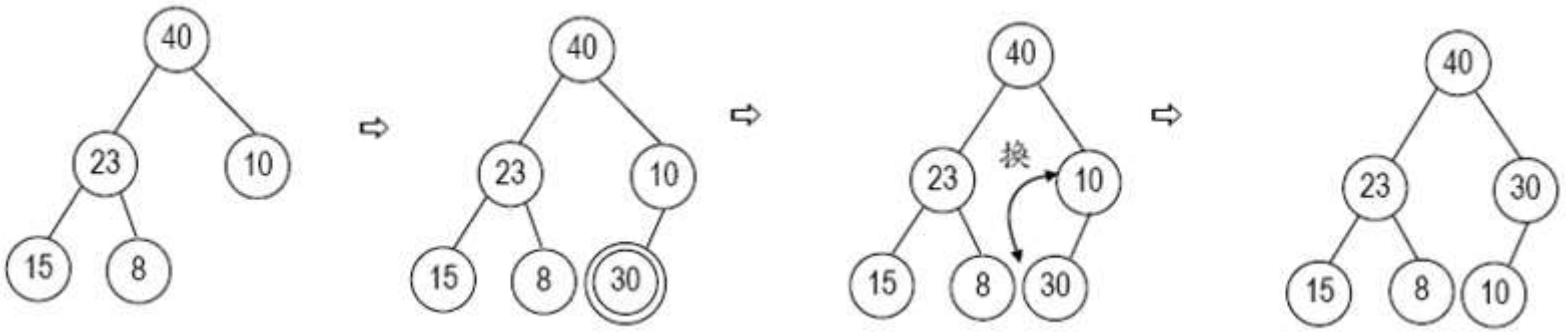
```
public class MaxHeap {
    int[] heap;
    int heapSize;
    int capacity;
    MaxHeap(int theCapacity) throws Exception{
        if(theCapacity<1) throw new Exception("Capacity需大於0");
        this.capacity = theCapacity;
        this.heapSize = 0;
        heap = new int[theCapacity+1]; // heap[0] is not used
    }
    Public Boolean isFull(){…}
    Public Boolean isEmpty(){…}
    Public void push(TreeNode newNode){…}
    Public void pop(){…}


}
```

Max Heap ADT implementation (2)
isEmpty, isFull

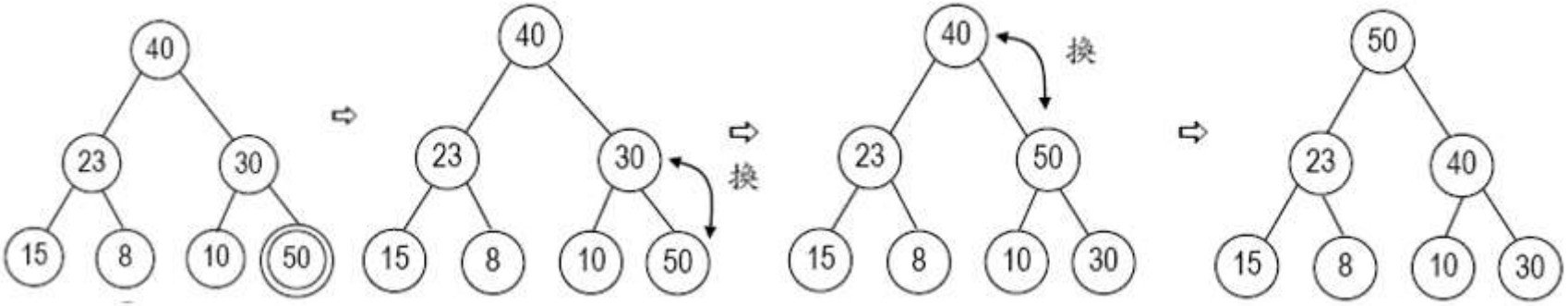
```
public boolean isEmpty(){
    if(this.heapSize == 0)
        return true;
    else
        return false;
}
```

```
public boolean isFull(){
    if(this.heapSize==this.capacity)
        return true;
    else
        return false;
}
```

push(int newValue) (1)



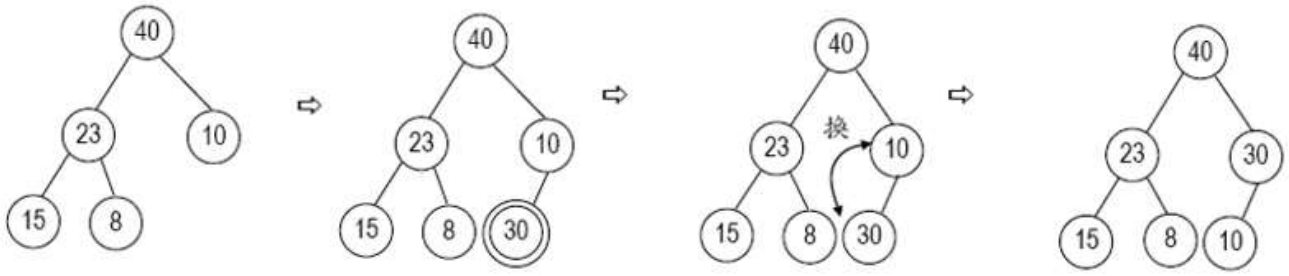
加入30



加入50

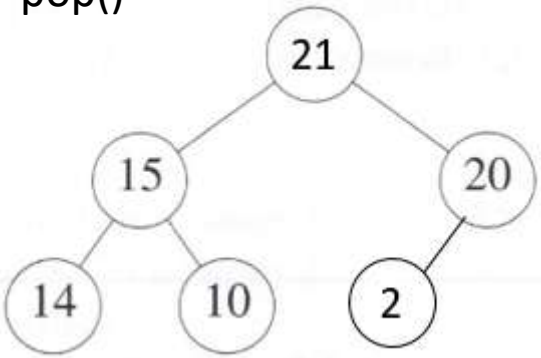
push(int newValue)(2)

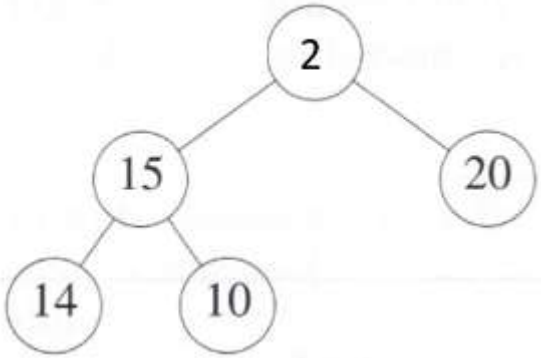
```
public void push(int newValue){
    if(isFull()){
        System.out.println("目前Heap已滿，不可push物件!");
        return;
    }
    heapSize = heapSize +1;
    int curIndex = heapSize;
    while(curIndex!=1&&heap[curIndex/2]<newValue){
        heap[curIndex] = heap[curIndex/2];
        curIndex=curIndex/2;
    }
    heap[curIndex] = newValue;
}
```

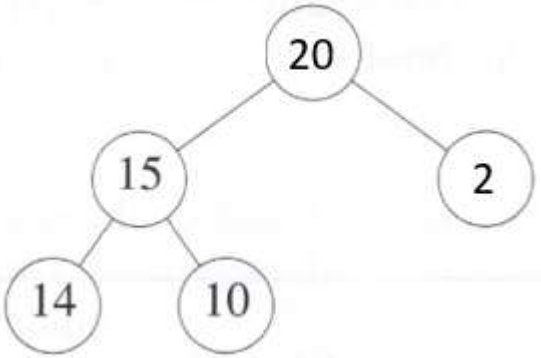


pop() (1)

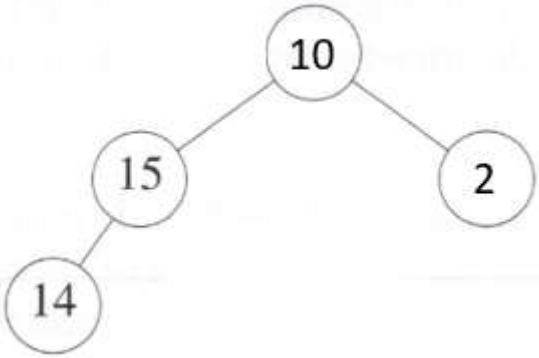
pop()

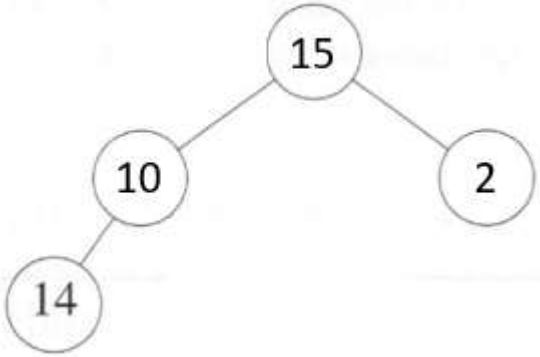

21
15
20
14
10
2

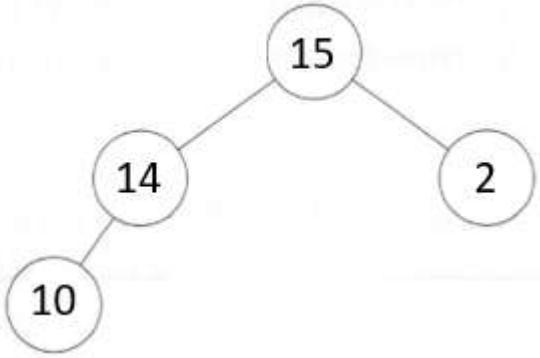

2
15
20
14
10


20
15
2
14
10

pop()

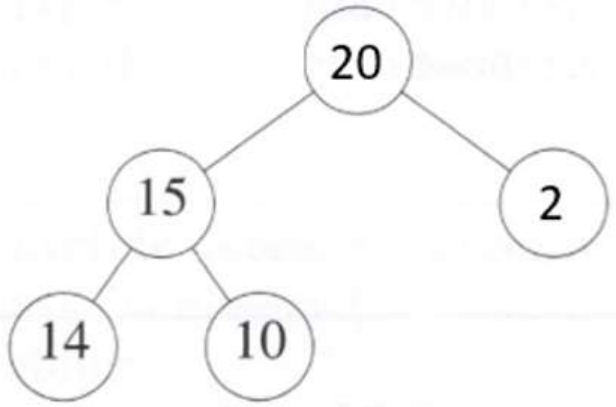

10
15
2
14


15
10
2
14


15
14
2
10

pop() (2)

```
public void pop(){
    if(isEmpty()) {
        System.out.println("目前Heap是空的，不可pop物件!");
        return;
    }
    int lastNode = heap[heapSize];
    heapSize=heapSize-1;
    int curNode = 1;
    int child =2;
    while(child<=heapSize){
        if(child<heapSize&&heap[child]<heap[child+1])
                child=child+1;
        if(lastNode>=heap[child])
                break;
        heap[curNode] = heap[child];
        curNode=child;
        child=child*2;
    }
    heap[curNode] = lastNode;
}
```



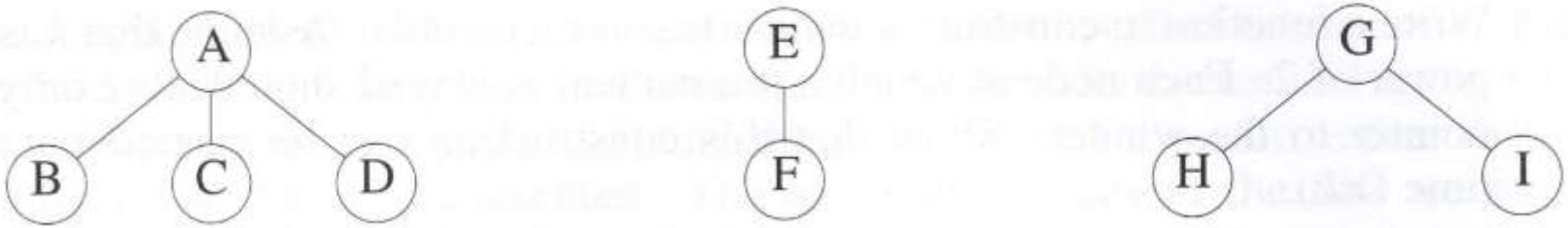
作業05-09

- 請實作Max Heap ADT

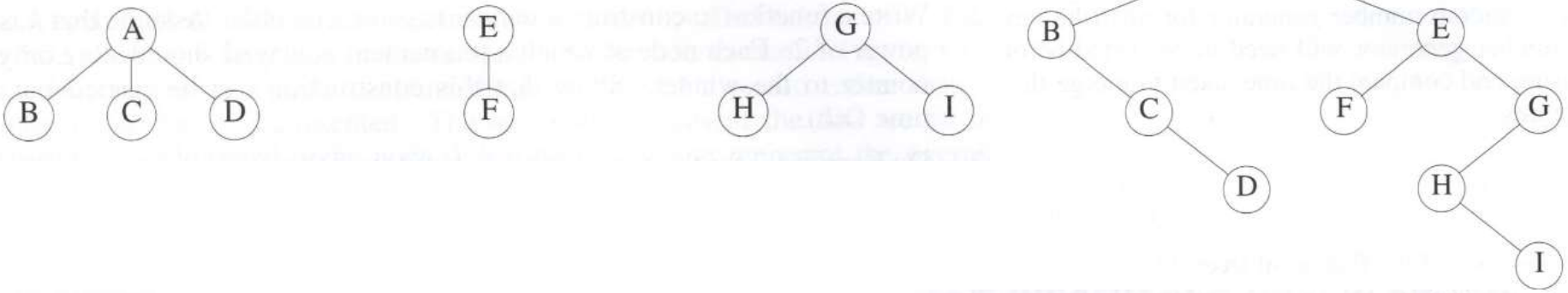
```
public class MaxHeap {
    int[] heap;
    int heapSize;
    //current size of max heap
    int capacity;
    //Maximum allowable size of heap
    MaxHeap(int theCapacity);
    //Create an empty heap that can hold a
    maximum of capacity elements
    public boolean isFull();
    public boolean isEmpty();
    public void push(int newValue);
    //insert a item into the heap.
    public void pop();
    //remove the largest item from the heap
}
```

樹林(Forest)

- 定義：一座樹林是指n(n≥0)棵非交集的樹所構成之集合。
- 下圖為三棵樹的樹林



轉換樹林為二元樹(1)

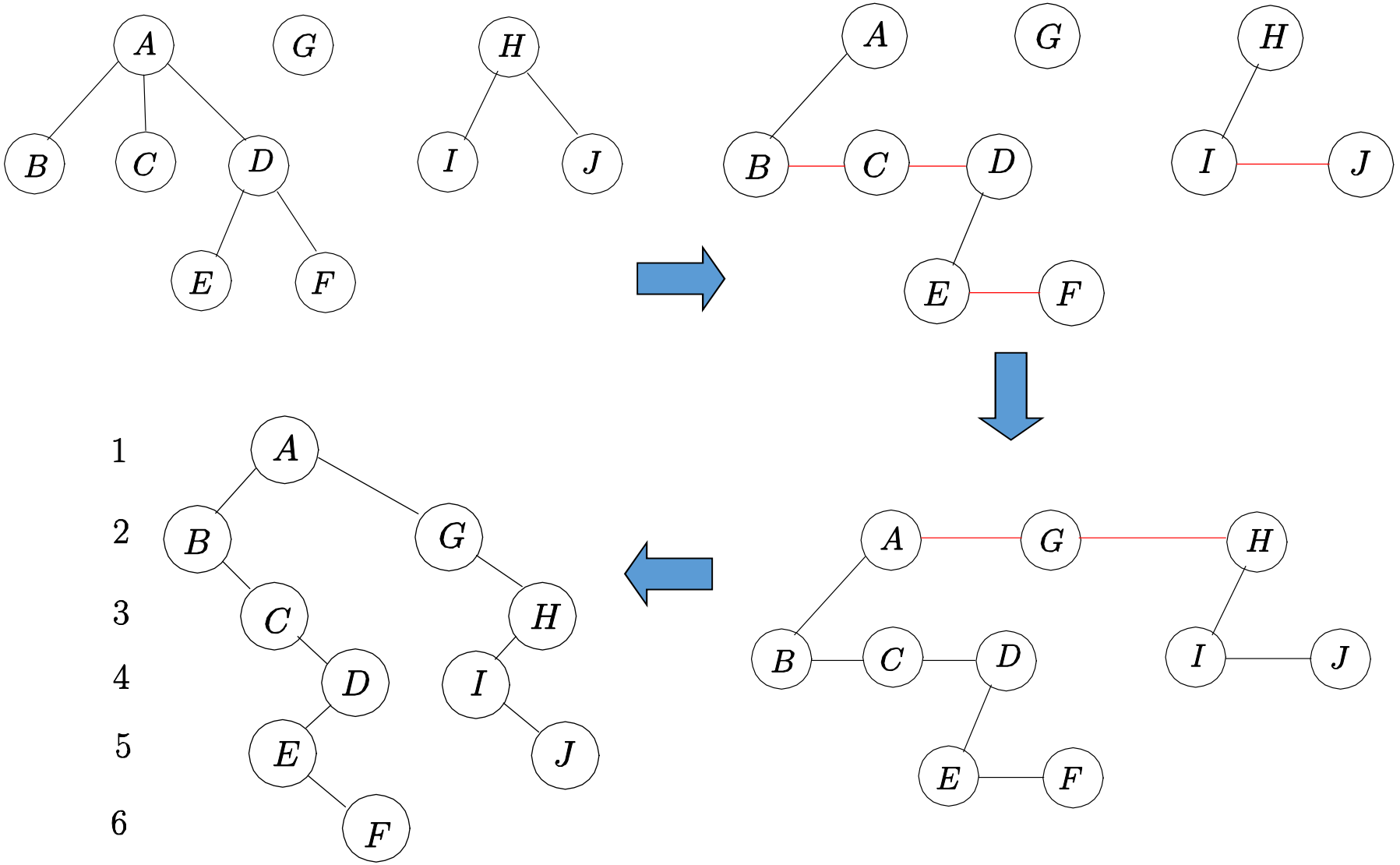


Definition: If $T_1, \cdots, T_n$ is a forest of trees, then the binary tree corresponding to this forest, denoted by $B(T_1, \cdots, T_n)$,

(1) is empty if $n = 0$

(2) has root equal to root (T_1); has left subtree equal to $B(T_{11}, T_{12}, \cdots, T_{1m})$, where $T_{11}, \cdots, T_{1m}$ are the subtrees of root(T_1); and has right subtree $B(T_2, \cdots, T_n)$. □

轉換樹林為二元樹(2)



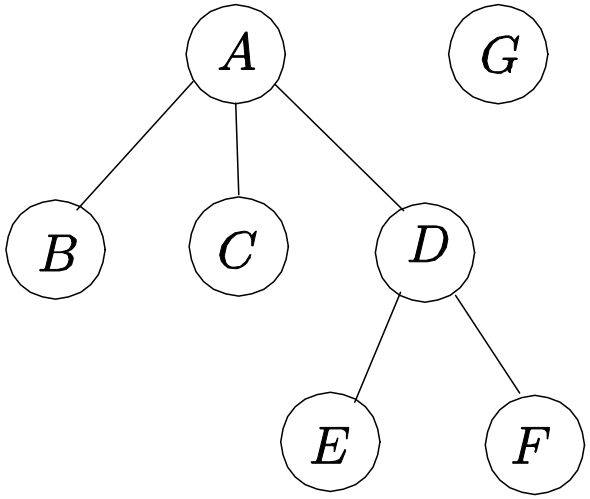
樹林的追蹤法

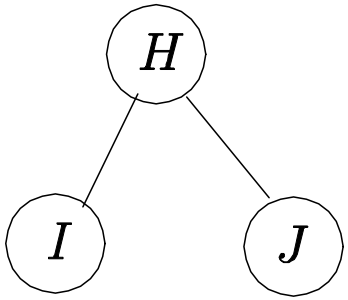
假設有 T1, T2, . . . , Tn 等n棵樹, 追蹤法有三:

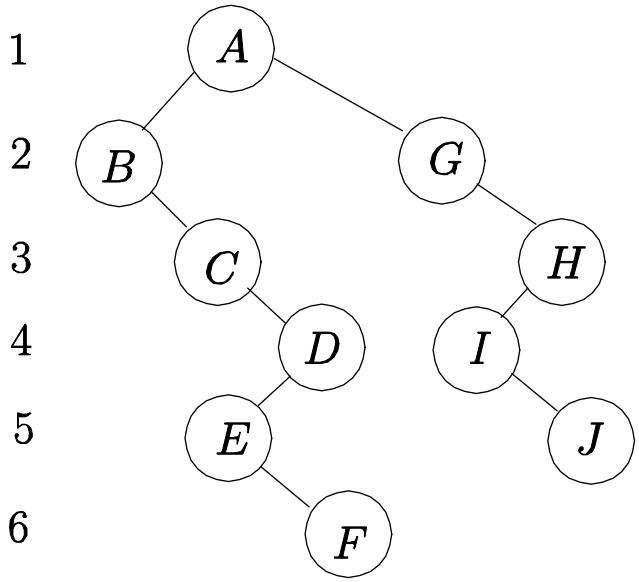
1. 中序追蹤:
   (1) 以中序法追蹤T1的子樹群。
   (2) 拜訪T1的樹根。
   (3) 以中序法追蹤T2, T3, …, Tn。
2. 前序追蹤:
   (1) 拜訪T1的樹根。
   (2) 以前序法追蹤T1的子樹群。
   (3) 以前序法追蹤T2, T3, …, Tn。
3. 後序追蹤:
   (1) 以後序法追蹤T1的子樹群。
   (2) 以後序法追蹤T2, T3, …, Tn。
   (3) 拜訪T1的樹根。

例題：樹林的二元樹表示法與追蹤

1. 中序追蹤：B C E F D A G I J H
2. 前序追蹤：A B C D E F G H I J
3. 後序追蹤：F E D C B J I H G A







練習

1. 中序追蹤: BCDAFEHIG
2. 前序追蹤: ABCDEFGHI
3. 後序追蹤: DCBFIHGEA