# 資料結構 HW2

1. Insertion:

   先檢查樹是否是空的。如果是空的則插入一 node 為 root，將 root 與 head 和 tail 之間的 thread 連接好並結束。如果不是，則搜尋插入點的位置。新增一 node。

   如果插入點為左節點，則將 parent node 的 left thread 給新增的 node 的 left，而原本 parent node 的 left 則接到新增的 node，新增 node 的 right 則以 thread 方式接到 parent node，如果 parent node 被 head 接上，因為新增 node 比 parent node 小，所以 head 改接上新增 node。

   如果插入點為右節點，則將 parent node 的 right thread 給新增的 node 的 right，而原本 parent node 的 right 則接到新增的 node，新增 node 的 left 則以 thread 方式接到 parent node，如果 parent node 被 tail 接上，因為新增 node 比 parent node 大，所以 tail 改接上新增 node。

2. Deletion:

   先檢查樹上是否只剩 root。如果是則刪除 root 節點並將所有變數設成初始化的值。如果不是則找出要刪除的 node 的位置。接下來以迴圈的方式重複找出該節點的 predecessor 或是 successor，並將找到的點的 number 取代原本的節點的 number，類似重複刪除節點的方式將要刪除的 node 傳到葉節點，因爲完全沒有改動中間樹的結構，所以完全不用維護任何 thread 或指標改動。接下來刪除 node。

   如果刪除點為左節點，則將刪除 node 的 left thread 給 parent node 的 left，如果刪除 node 被 head 接上，則讓 head 改接 parent node。

   如果刪除點為右節點，則將刪除 node 的 right thread 給 parent node 的 right，如果刪除 node 被 tail 接上，則讓 tail 改接 parent node。

3. Inorder_run:

   先從 head->right 找出第一個要走的 node。接下來以迴圈執行，當還沒走到 tail 時，輸出現在走到的 node 的值，再往該 node 的 successor 走。

   如果該 node 的 right 是 thread，node->right 就是 successor。

   如果該 node 的 right 不是 thread，則從 node->right 開始往左走到葉節點，該點就是 successor。

4. Reverseorder_run:

   先從 tail->left 找出第一個要走的 node。接下來以迴圈執行，當還沒走到 head 時，輸出現在走到的 node 的值，再往該 node 的 predecessor 走。

   如果該 node 的 left 是 thread，node->left 就是 predecessor。

如果該 node 的 left 不是 thread，則從 node->left 開始往右走到葉節點，該點就是 predecessor。