# DsNP HW 5 Report

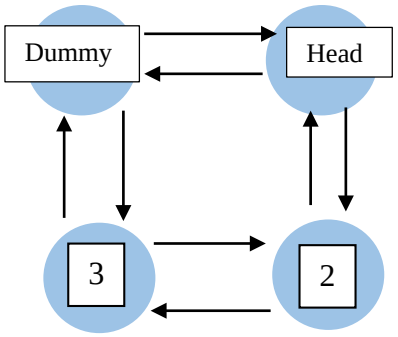
電機三 B04901069 林志皓

（一）三種 data structure 簡介及實作方式

## 【 Doubly Linked List 】

**1. 簡介：**

如右圖所示，將一串資料串成一個環，每一個 node 都分別有指向前一個跟後一個資料的指標。



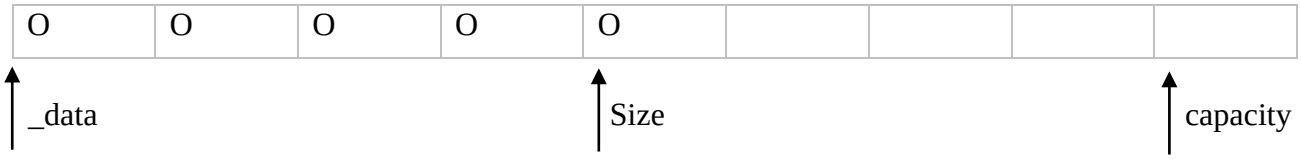

**2. 實作方式：**

額外創造一個 Dummy Node ，(當沒有任何資料時 Head 指向此)，之後在新增資料時保持它在 Head 之前，最後一筆資料之後，好處是可以當 iterator 指向 end 的位置，也讓 push_back 資料時能馬上找到該插入的位置，讓整體速度提升。 而 sort 的方式我是選擇用 Insertion Sort ，因為很好實行(code 少)，而比起同樣簡單的 BubbleSort 每次執行都需 $O(n^2)$的時間，此方式若在接近 sort 完的 case 可以很快執行完(O(n))。

## 【 Array 】

**1. 簡介：**

直接將資料放成一排(連續)，並不需要額外使用指標。



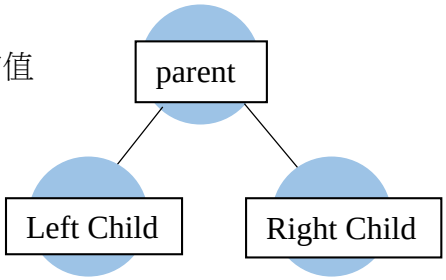

**2. 實作方式：**

_data 指向這些資料的第一筆的位置，有另兩個 data member，size 紀錄已儲存多少筆資料，capacity 則記錄目前所能容納的最大資料量，當滿了(size = capacity)，則向 heap memory 要一段兩倍 capacity 大小的記憶體，將原本的資料複製過來，然後將原本的 array 所用到的紀體還給系統 (因此 capacity 的大小都是2的指數倍 )。由於資料放在記憶體中連續的位置，因此可以很快取用(直接位移一定量值的記憶體位置而不需要指標指來指去) 。

## 【 Binary Search tree 】

**1. 簡介：**

如右圖所示，每一筆資料以 Binary Tree 的方式排列，而每一個 Node 的 Left Child 的值皆小於等於該 Node 的值，而 Right Child 的值皆大於該 Node 的值。



## 2. 實作方式:

由於 BST 的實行方式較為複雜，每個人的方式也相差甚遠，因此分為下面幾點說明:

1. 在每個 BSTreeNode 裡我用了這些 data member 及其原因 :
   (1) _data → 放儲存的資料
   (2) _state ( 這個 node 是 root、LeftChild 還是 RightChild)
   →利於 iterator 找下一個或上一個 Node
   (3) bool _isEnd : 判斷此是否為 EndNode，可以當作 iterator 的 end()，判斷是否為 empty()
   (4) 三個 pointer: _parent 、_left、_right 分別指向 adjacent Node
2. 我設定了 一個 EndNode ( _isEnd = true ，其他的 Node 都是 false )，在 insert 的時候令此 node 值為最大(遇到他就放到他的 left)，這樣 iterator 在 traverse 時最後才會走到他
3. Iterator 在執行 ++ 或是 - -時，我依據每個 node 是 Left/Right Child 來找到他的下一個或上一個 Node (詳情請見 bst.h) → 個人認為這樣對我來說比較直觀，容易實行，但是比起去 keep a trace，此方法在 traverse 的時候會稍慢一些些 (adtprint 時)
4. Class iterator 中我設了兩個 function 去找任一個 subtree 的 最小/大的 node，利於 traverse
5. 我自己覺得除了 traverse 之外最大的困難就是 delete Node，我在實行的時候分為4種狀況:
   (1) leaf(no children) →直接刪掉
   (2) have left child but no right child →把 subtree 直接接到他的 parent
   (3) have right child but no left child →同上
   (4) have left child and right child
   → 用上一個(或無則下一個)node 的值取代，然後 delete 該 用來取代的 node

（二）三種 data structure 實作成效與比較 (單位 : 秒 )

| | Dlist | array | BST |
|---|---|---|---|
| **adta –r 100** | 0 | 0 | 0 |
| **adta –r 1000** | 0 | 0 | 0 |
| **adta -r 10000** | 0 | 0.02 | 0.04 |
| **adta –r 100000** | 0.04 | 0.03 | 0.18 |
| **adtp** | 0.04 | 0.05 | 0.05 |
| **adts** | 155.7 | 0.09 | 0 |
| **adtd –f 5000** | 0 | 0 | 0 |
| **adtd –b 5000** | 31.98 | 0 | 0 |
| **adtd –r 5000** | 2.09 | 0 | 29.65 |
| **adta –s @@@@@ adts adtd –s @@@@@** | 0.01 | 0.06 | 0 |
| **adtd -a** | 0 | 0 | 0.02 |
| **Total time used** | 189.8 | 0.25 | 29.94 |
| **Total memory used** | 6.129 M Bytes | 5.62 M Bytes | 7.895 M Bytes |

## 觀察：

1.在放進新的 data 時，可以發現 BST 隨著資料的增加，新增資料所需的時間也越多，因為他在放進去的同時就要把該資料逐一比較，放到適當的位置。
2.在 print 時三種資料結構 performance 差不多
3.而在進行排序時，三種之間的差異相當明顯，用 InsertionSort 的 Dlist 花了將近三分鐘，使用內建排序(大概是 MergeSort 或是 QuickSort)的 Array 只需要 0.09秒 ，而 BST 所有資料已經排好，因此不須再花時間
4.在 adtd –front 三者表現一樣，但是 adtd –back 時 Dlist 花費比其他兩者多很多的時間，推測是因為要找到 end 需要使用指標去找，因此在資料量大時花費較長時間；而 adtd –random 則是 BST 花費最多時間，推測是因為 BST 要找到指定位置時進行 traversal 需要較長時間來讓 iterator 找到下一個 node 的位置；而如果是找指定值 (adtd –s @@@@@)則 BST 因為它的樹狀結構及已經排序好的特性，可以用最快的時間完成。
5.Array 幾乎不需要額外的 data member 去紀錄 node 的資訊，所花的記憶體最少；而 BST 用了最多 data member，因此用了最多記憶體，但三者之間的差異並沒有執行時間顯著。

## 結論：

在實行這三種資料結構時，個人認為 array 是最容易，performance 整體而言也是最好的，是一個 CP 值最高的資料結構 ； Dlist 稍複雜一些，但幾乎沒有勝於其他兩者的地方，是個適合成為教科書中成為歷史的資料結構；而 BST 最有挑戰性，在實行的過程其實相當有趣，要想許多方式去 handle 各種狀況，如果是在需要經常排序的 case，用 BST 是最好的選擇。