2019DSnP-HW5 實驗報告

電機二范育瑋

1. 程式架構

由於 Dynamic Array 與 Doubly Linked List 都是寫好的 header 讓我們填空，因此只會就新增的 private helper function 或一些特殊的實做方法做說明。另外這次三種 ADT 我都沒有選擇使用 _isSorted。

- Dynamic Array

  expand():
  由 push_back()呼叫，在 array 空間不夠時，以 0,1,2,..,2^n 擴增_capacity，它會 new 出一塊更大的空間，並將原本的資料複製進去。

- Doubly Linked List

  bubble_sort():
  由 sort()呼叫，即對_head 所接起來的元素做排序，其中在交換元素時，只需要將_node 內的 _data 互換就好，不需要把整個_node 交換。

  swap(a, b):
  由 bubble_sort()呼叫，交換 a 和 b，在這個 class 內就是 A node 的_data 與 B node 的 _data。

- Binary Search Tree

  BSTreeNode:

  這個 class 裡面我放了 4 個 private data member 分別為
  1.T _data : 存資料
  2.BSTreeNode* _child[2]： _child[0]=_left child , _child[1] = right child
  3.BSTreeNode* _parent : 存父節點
  4.int _dir : 代表父節點是走左(0)/右(0)到當前節點，其中_root 的_dir = -1
  之所以沒有照著用_left 與_right 存左右子節點，是因為在實做 Tree Traversal 時發現在要更新_parent 的_right 與_left 指標時，非常冗長，這樣的作法可以直接透過 _dir 取得相對應方向的子節點，程式碼乾淨許多，效能也有提昇。

  static predecessor(node)＆successor(node):
  在新增刪除與 tree traversal 時被呼叫，回傳一個 BSTreeNode*，其中設為 static 是因為它沒有什麼理由需要 this pointer，設成 static 後還能夠被 iterator 這個 class 呼叫，predecessor 回傳左子樹的最大 node，successor 回傳右子樹的最小 node。

delete_node(node):

由 erase()，pop_front()，pop_back()，呼叫，它會回傳一個刪除成功與否的 bool，並刪除指定的 node。

clear_tree():

由 clear()呼叫，以左-右-中 PostOrder 的順序刪除子樹與_root。

2. 實驗分析比較

- 測資：

下表為針對各個資料結構做同樣的操作所需要的時間以及與 ref 程式 Normalize 後的數字。

	Array	Linked List	BST
do.adta	1.78 / -0.04	0.58 / -0.18	18.35 / -0.05
do.adtd	0.01 / 0	5.51 / 0.16	42.78 / 0.62
do.adtd_f	0 / 0	0.02 / 1	0.12 / -0.29
do.adtd_b	0 / 0	0.03 / 0.5	0.15 / -0.17
do.adtq	109.3 / 0.12	106.7 / -0.07	6.4 / -0.02

單位(s)

do.adta：隨機增加 10000000 比資料
do.adtd：自 100000 比資料，隨機刪除 30000 個
do.adtd_f：自 100000 比資料，用 pop_front()刪光 100000 個。
do.adtd_b：同上，用 pop_back()刪。
do.adtq：自 100000 比資料中，隨機尋訪 100000 個隨機字串是否存在。

其中在生成 do.adtq 時，將字串長度設為 4，確保用來搜尋隨機生成的字串對原資料內字串的覆蓋率不會太低。

- 實驗分析

由實驗數據可以發現，在新增資料方面，BST 的效率遠低於 Array 與 Linked List，因為它需要將資料保持排序好的狀態；就刪除內部資料的操作，同樣的由於 BST 要維持樹的順序，效率也不如前兩者；然而尋訪資料部份，由於 BST 的特性，尋訪的複雜度為 O(height)，其中 height 在大多數情況會比 n 小的多，與實驗相符。

此外，與 ref 程式比較的結果可以發現，在 Doubly Linked List 的 delete_front 與 delete_back 看似特別慢，猜想是由於這樣的測資太小，導致在位數取估計時造成資料失真；BST 的 delete 也是另一個相對範例程式慢的操作，或許是我把 trace 都存在 node 導致記憶體用量大，拖慢整體效能。

- 結論

若要操作的資料的順序非常重要，而且不需要頻繁新增、刪除，需要大量的尋訪，Binary Search Tree 是比較理想的資料結構。
反之，需要頻繁新增、刪除時，Array 與 Linked List 是比較好的資料結構，其中由上表可以發現，由於 Array 的資料結構比較簡單，需要的操作少，在大多數情況比 Linked List 的效率好。