資料結構與程式設計 HW5 Abstract Data Type

b07901021 電機二 潘世軒

1.各資料結構的比較

	Array	**Dlist**	**BSTree**
插入	從末端(破壞排序)	從任意位子 (破壞排序)	依大小排序， 進入適合的位子
存取	任意位子	從頭開始	Binary search
比較快的部份	插入、刪除、存取	插入、刪除、存取	排序
比較慢的部份	排序	排序	需要移動到 iterator 的部份：插入、刪除

2.各 function 的實作(iterator 只說明++，--可類推，其他功能比較 trivial)

	Array	**Dlist**	**BSTree**
Iterator++	_node++	_node=_node->_next	如果右邊有東西的話，尋找右支的最小值回傳並紀錄 trace。 如果右支沒有東西的話，往上找直到出現向左岔路，回傳該 iterator、紀錄 trace。
begin()	Return 第一筆資料的 iterator		
end()	Return 最後一筆資料的「下一個」iterator		
empty()	用_size 來判斷	如果 head 的前後都沒有東西(尚未新增元素進去)，可判斷為空	用_size 來判斷 PS:在 BSTree 的部份，因為要去數有幾個成員有點太麻煩，因此選擇直接用_size 來記大小
size()	Return _size	用 iterator go through 整個 dlist，並計數	Return _size
operator[]	Return _data[i]	X	X

	Array	Dlist	BSTree
push_back()	如果 array 要爆了就先 expand(把 capacity 變成兩倍)，之後把新增元素放到最後一個	把新增元素放到最末並更新相關的_prev 和 _next	X
insert()	X	X	根據數據大小從_root 開始走，決定位置後 new 一個 node 給新增的元素，並確保他的 parent 有指向他
pop_front()	把最後一個數據移到第一個，並更新_size	讓_head 往前一個，更新 dummy node，然後刪除_head	erase(begin())
pop_back()	_size--	Dummy 變成_head，最後一個變成 dummy，然後刪除 _head	erase(--end())
erase(iter)	把最後一個數據移到欲刪除的位置，並更新 _size	將欲刪除的前後互相連接，並刪除要刪除的	尋找 successor，並將 successor 移到欲刪除的位置，如果沒找到的話，將 parent 跳過玉刪除的和下一個連接起來。更新_size
find()	透過 iterator go through 整個資料結構並 return 相符的 iterator		從_root 開始進行 binary search，「並紀錄 trace」， return 相符的 iterator

	Array	Dlist	BSTree
clear()	_size = 0	透過 iterator go through 整個資料結構並一一刪除，重置 _head	將_root 的左半邊用 pop_front()刪掉，再將_root 的右半邊用 pop_back()刪掉，不刪除_root，最後_size = 0 PS:不將_root 刪掉的原因是我在 BSTree 的 constructer 的時候就已經先把_root new 出來了，為了保持一致所以這麼做
sort()	::sort	Bubble sort	直接 return
print() **(=verbose)**	X	X	透過一個 static int 來儲存現在所在的 level，並透過遞迴的方式印出樹

3.實驗設計(O3 編譯)

adta -r 100000
adts
adtd -r 10000
adtd -f 10000
adtd -b 10000
adtr 4

adta -r 1000000
adtd -r 10000
adtd -f 10000
adtd -b 10000
q -f

設計說明：(皆使用 random 模式測效能)包含插入，排序，刪除等等試驗，希望藉此判斷三種資料結構的差異性，其中刪除包括隨機，從前面，和從後面刪除，然後測試 reset 長度功能，並透過規模相差 10 倍的資料量來比較資料量大小時的差異。

PS：第二部分本來想要都刪除 100000 個，但是我本來試試看刪除 100000 個的時候，睡了個午覺起來發現 BSTree 還沒跑完，所以作罷 QQ

4.實驗結果(我的 code/refcode)(將時間相較花比較多的標起來)

	Array	Dlist	BSTree
adta -r 100000	5.188/**5.574** (MBytes)	5.672/**5.203** (MBytes)	5.598/**5.68** (MBytes)
	0.07/**0.05**(s)	0.01/**0.02**(s)	0.04/**0.17**(s)
adts	0.08/**0.07**(s)	107.2/**143.1**(s)	0.00/**0.00**(s)
adtd -r 10000	0.00/**0.00**(s)	4.41/**3.19**(s)	23.48/**30.27**(s)
adtd -f 10000	0.00/**0.00**(s)	0.00/**0.00**(s)	0.01/**0.00**(s)
adtd -b 10000	0.00/**0.00**(s)	0.00/**0.00**(s)	0.03/**0.00**(s)
adtr 4	0.00/**0.00**(s)	0.00/**0.00**(s)	0.1/**0.00**(s)
adta -r 1000000	47.48/47.86 (MBytes)	60.7/**60.23** (MBytes)	60.76/**60.71** (MBytes)
	0.17/**0.3**(s)	0.14/**0.08**(s)	1.69/**2.25**(s)
adtd -r 10000	0.00/**0.00**(s)	62.82/**49.27**(s)	720/**745**(s)
adtd -f 10000	0.00/**0.00**(s)	0.01/**0.00**(s)	0.02/**0.00**(s)
adtd -b 10000	0.00/**0.00**(s)	0.00/**0.00**(s)	0.03/**0.01**(s)
q -f			

結果分析：

對 Dlist 來說，主要時間會花在 sort 和隨機刪除的部分，sort 使用 bubble sort，使用 quick sort 可能可以更快一點，而隨機刪除是因為要還要把前後連結起來，相較 array 會花比較多時間，相較於 array 卻有其他好處(見 array)。

而 BSTree 則是把時間花在隨機刪除的部分，正如上面分析，由於 BSTree 的 iterator 的 code 比較複雜，而 BSTree 的隨機刪除由於需要去找 successor 的關係，需要有很大量的 iterator 運算，因此在隨機刪除這個部分就花了超級超級多的時間(還以為電腦當機)，但是 BSTree 因為在放入資料時就已經排序，所以完全不需要花時間來排序，優於其他兩者。

雖然在圖表中 array 的速度和記憶體用量明顯都優於另外兩者，但是實際上 array 還是有些許的問題，比如說在某個特定的地方如果要插入數據的話其實會很麻煩(本次作業並沒有要完成 array 的 insert)，相比，dlist 只要動個指標就好，再者，本次作業中的 popfront，array 是把最後一個搬來，但直覺應該是要把所有資料平移才對(也才不會破壞排序)，但這樣就必須花大量的時間在搬動資料，若是資料總數一直變動，也需要花許多時間在新增記憶體的搬動資料過程。但是 array 省記憶體卻是無庸置疑的，因為不需要用 pointer 去存一些記憶體位址。

總而言之，三種資料結構各有其優缺點，要對應不同的情況選擇適合的來使用(比方說，如果存一個資料庫的話，就很適合用 BStree 來實行(有點像目錄那樣(?)))