Performance study report

B03901179 溫明浩

資料結構實作

Array:

利用_size 儲存目前大小，_capacity 儲存總共可用空間，當空間不足時，就會申請更大空間（實作為兩倍），將舊資料整個複製過去，再把舊空間釋放。因此，資料在記憶體裡永遠是連續的，iterator 在移動時可以直接指到下一塊記憶體，對於每筆資料可以輕易取得位置，end()也很直觀。

Dlist:

資料為不連續，每個 node 會有_next & _prev pointer 去紀錄前後資料位置，因此 iterator 在移動時需要透過此二 pointer 去得到下一個 node 的位置。另外，會設置 dummy node 串聯頭尾，方便取得 begin、end。

在 sorting 方面會花較多時間找資料。為避免多次查找，在此選擇將資料全部複製到新開的 array，透過 STL 的 sort()，最後再放回去，不但 code 簡潔容易，速度上也比 bubble, insertion 快很多，但是會花費額外的記憶體空間。

Bst:

每個 node 會儲存_lchild、_rchild、_parent 的位置，BSTree 存放_root、_size。使用_parent pointer 在於能夠往上追蹤，而非單向往下。

新增資料時從_root 開始，大於的往右存放，小於等於則往左，找到末端點就利用 pointer 連結。刪除資料時，分成三種:

沒有 children：只要刪除即可

一個 child：將小孩與上面的父母連結後再刪除

兩個 child：向下尋找 next larger 的 node ，用他取代即將被刪除的 node

另外，如果刪除的為 root，也要去更新它

iterator++：向下尋找 next larger node，如果搜尋不到，則向上找 parent，如果也為 NULL 則代表為最大，回傳 NULL

iterator--： 向下尋找 next smaller node，如果搜尋不到，則向上找 parent。

iterator 除了有_node 之外，還有一個_getRoot，當呼叫 end 時，會把_root 傳入，之後再處理一時，利用它來尋找最大值。

實驗設計：

test 1:小量資料增減 +5000、-1000、+7000、-5000、+5000

	Array	Dlist	Bst
Time(second)	0	0	0
Memory Used(Mb)	0.3771	0.4492	0.4648

test 2:中量資料增減 +10000、-5000、+30000、-25000、+50000

	Array	Dlist	Bst
Time(second)	0.03	0.01	0.04
Memory Used(Mb)	2.186	3.34	3.281

test 3:大量資料增減 +100000、-50000、+300000、-250000、+500000

	Array	Dlist	Bst
Time(second)	0.45	0.16	1.06

Memory Used(Mb)	47.94	36.29	36.27

test 4:巨量資料 sorting +1000000
括號內為未 sorting 的時間消耗量

	Array	Dlist	Bst
Time(second)	0.87(0.51)	0.78(0.17)	1.14
Memory Used(Mb)	48.01	91.29	60.78

Test 5 巨量資料 print +1000000

	Array	Dlist	Bst
Time(second)	0.9	0.66	1.89
Memory Used(Mb)	47.87	60.8	60.68

在小量資料時，三種方式的存取都差不多
但是到了大資料量時，因為 Array 要一直複製刪除，而增加不少時間，而 bst 則在 insert 部份就花了相當時間，Dlist 因為只需要串聯起來，故花費最少時間
在 sorting 的部份，Dlist 在 sort 的複雜度並非為 n^2，雖然速度上稍微輸給 array(因為要將資料取出再放入)，但是整體還是比較快的。
由於 Dlist 兩個 pointer，而資料容量小於 8byte，因此 Dlist 所佔記憶體還是比較大
不過再 print 時，因為 iterator 複雜度的關係，因此 array 還是比較快的，而 bst 則因為要一直繞，所以速度上比較慢。