HW5 Report

B06901144
電機三 吳悠嘉

一、 實作

1. 三種資料結構比較

	DList	Array	BST
資料位置	分散，每個 node 前後連接	連續的分布	分散，node 以指標連接
iterator	照著資料 linked 的順序，++往 node 的 next 走，--往 prev 走	照記憶體空間的順序	Inorder traversal ++指到 node 的 successor，--指到 predecessor
插入資料	接在最後一個資料	需檢查記憶體空間，已滿時要重新配置	從 root 開始與所經 node 比大小，較大則往右，較小往左
刪除資料	將欲刪除之資料的前後節點相互連接	把在最後的資料補至欲刪除資料的位置	有一 child 則直接將 child 補上刪除 node 的位置。若有兩個，則將其 successor 補上。

2. 程式設計

(1) Dlist

- 在 class Dlist 的 private 新增變數：
  DListNode<T>* _dum : 指向 Dlist 的 dummy node
  DListNode<T>* _last : 指向最後一筆資料
- empty() :
  若_head 的_next 指到_head，代表無其他 node，此時回傳 true，否則回傳 false。
- size() :
  先將 iterator 指到 begin()，每移動一次記數一次，直到 iterator 指到 end()為止。
- push_back(x) :
  若插入的資料為第一筆資料，將_dum 指到_head，再將

_head 與_last 指到插入的資料，此時_dum 和此資料互為_next 和_prev。
若 list 已有資料存在，則將_last 指到新插入資料，更新此資料之前後位置的_node 以及自身的_prev 與_next。
無論哪種操作，結束後都將_isSorted 設定為 false。

- pop_front():
  在非 empty 的狀態，將_dum 的_next 指到_head 的_next，再刪除原位於_head 的 node，並更新_head 至_dum->_next
- pop_back() :
  與 pop_front 相似。在非 empty 的狀態，將_dum 的_prev 指到_last 的_prev，刪除原為於_last 的資料，並更新_last 至_last->_prev。
- erase(pos) :
  若 empty 回傳 false。若否，則刪除位於 pos 的資料，並將 pos 前後的 node 相互連接，若 pos 為_head(_last)，將_head(_last)指向_head->_next(_last->_prev)，回傳 true。
- find(x) :
  將 iterator 由 begin()指到 end()，若找到與 x 相等的資料，則回傳其所在位置，若直到 end()都找不到則回傳 end()。
- erase(x) :
  先利用 find(x)尋找 x，若 find(x)回傳 end()，則回傳 false，若否則回傳 erase(pos)。
- clear() :
  不斷 pop_front 直到資料全清空，並將_isSorted 改成 false
- sort() :
  使用 Quicksort。
  在 class Dlist 的 private 建立三個 function：
  >Partition(把比指定數字小的數往前放，較大的往後放)
  >QuickSort、
  >compare (用於比較輸入的 iterator 誰的位置在前，誰的在後，若前者在前則回傳 true，在後則回傳 false)。
  sort() 會先呼叫 QuickSort(begin(),--end())，在開始 partition 之前會先利用 compare 比較輸入的 iterator 的正當性，若 compare 為 true，則進行 partion，並利用 partion 所得的 iterator q，進行 recursive 的操作：
  QuickSort(begin(),--q) 以及 QuickSort(++q,--end())
  以此類推。

（2）Array

- empty() :
  直接檢查_size。若_size = 0 則回傳 false。
- push_back(x) :
  若_size 和_capacity 相同，則須重新配置記憶體，若此時_capacity 為 0，則新增大小為 1 的記憶體；若大於 0 則新增為原_capacity 2 倍的記憶體，更新_capacity。將在原記憶體的資料複製到新記憶體，刪除原記憶體，並將新資料存入新記憶體中，++_size，_isSorted 更新為 false。
- pop_front() :
  若_size 為 1 則直接--_size，若_size 大於 2，則將最後一比資料補上，並--_size。
- pop_back() :
  直接--_size。
- erase(pos) :
  方式同 pop，將最後一個資料補到要刪除的位置，將_isSorted 改為 false。
- find(x) 、erase(x) :
  方式同 dlist。
- clear() :
  無須丟資料，只需--_size，並將_isSorted 改為 false。

（3）BST

- 在 BSTreeNode，每個 node 都有_childl (left child)，_childr (right child)，以及_parent，node 的資料存於 _data。
- Dummy node (_dum)的設計：
  _dum 為_root 的_parent，且_root 為_dum 的 left child。這樣的好處是，無論對 tree 做何種操作，都比較不會動到_dum，在設計 tree insert , delete 時不需再額外考慮_dum 的存在，且可確保 end()會永遠只到_dum，而非任一有意義之 node。初始化將_dum = _root(為整個樹的根)。
- TreeMin / FindMin(BSTreeNode<T>* _r) :
  尋找以_r 為根的 tree 擁有最小值之 node。做法是由_r 不斷往左走
- TreeMax / FindMax(BSTreeNode<T>* _r):
  想法同上，只是是在找最大值，由_r 不斷往右走。
- begin() :
  若為 empty 則回傳 end()，若非，則呼叫 FindMin(_root)，將 iterator 指到最小值。

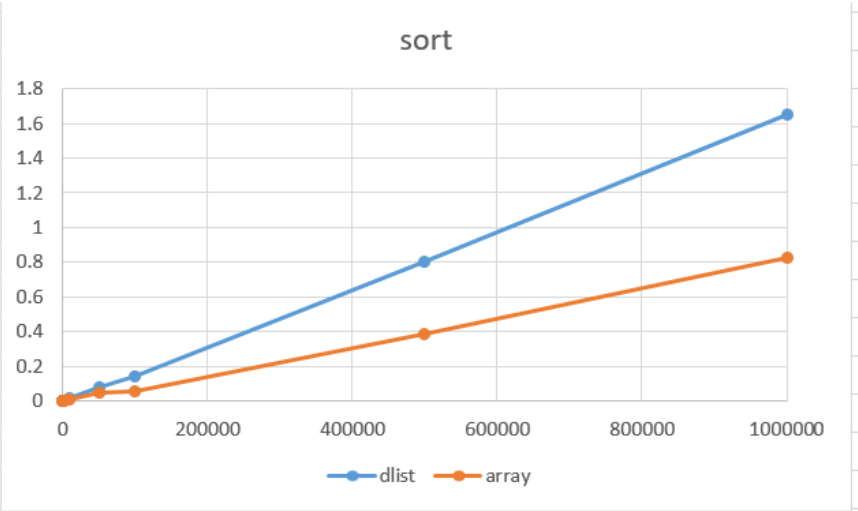
- end() :
  可直接回傳 iterator(_dum)
- empty() :
  若_root = _dum，則回傳 true。
- insert(z) :
  若為 empty，則新增一個 node，將_root 指向此 node，此 node 以_dum 為_parent 且為_dum 的 left child。
  若已有資料存在，則按照 BST 的規則，找出資料應放置的位置。將_size++。
- transplant(u,v) :
  將 u 節點以 v 節點取代，方便用於_delete 覆蓋掉欲_delete 的資料而做。
- _delete(pos)：
  位於 pos 的節點只有 1 個 child，直接呼叫 transplant(u,u->child)，將此 node 與他的小孩互換。
  若有兩個則用其 successor 覆蓋掉。
- pop_front/ pop_back :
  利用_delete，刪去掉 begin() / FindMan(_root)回傳之 node
- find (k) :
  利用 tree search，將 k 與遇到的 node 比大小，若較大則往右找，若較小則往左找，若相同則回傳其位置。沒找到則回傳 end()。
- erase :
  方式與前兩中資料結構類似概念，只是這次可利用已寫好之_delete。

二、 實驗

1. Sort (BST 已排序好，故不列入討論)

資料數量	時間(s)	
	dlist	array
10	0	0
100	0	0
1000	0	0
10000	0.02	0.01
50000	0.08	0.05
100000	0.14	0.06
500000	0.8	0.39
1000000	1.65	0.83

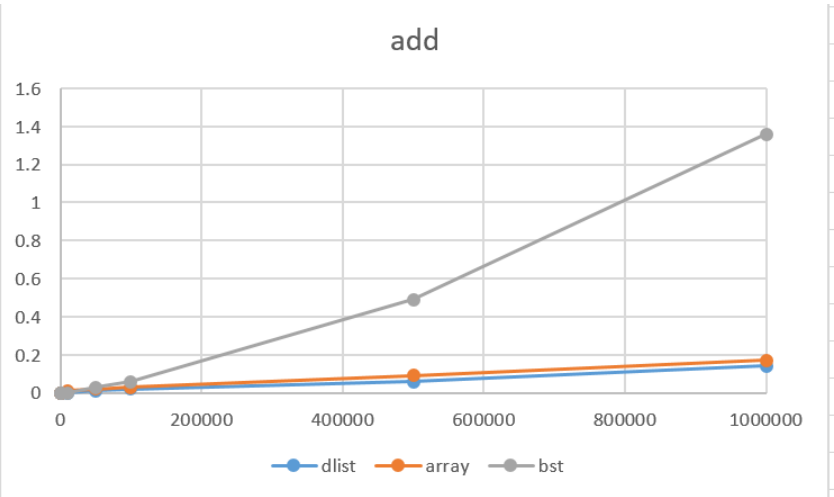
->資料數量小時差異不大，當資料數越大，array 的速度比 dlist 的快越多。



2. Add

資料數量	時間(s)		
	dlist	array	bst
10	0	0	0
100	0	0	0
1000	0	0	0
10000	0	0.01	0
50000	0.01	0.02	0.03
100000	0.02	0.03	0.06
500000	0.06	0.09	0.49
1000000	0.14	0.17	1.36

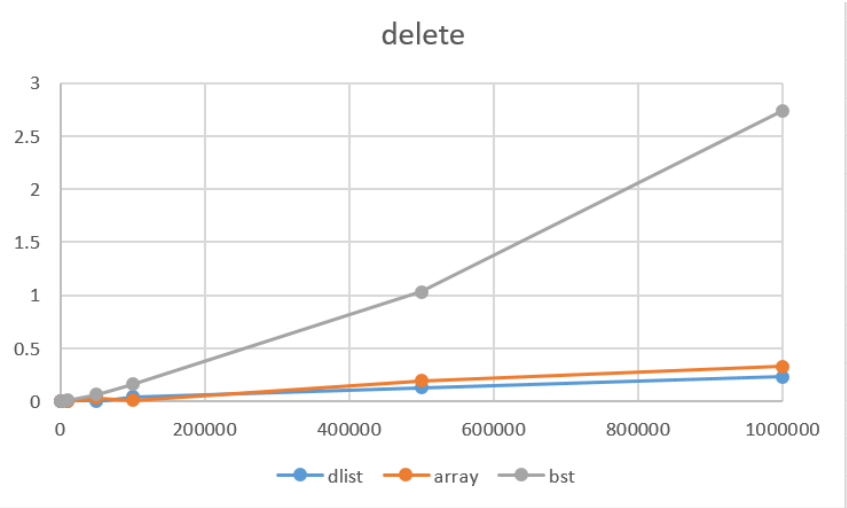
->資料數量小時差異不大，但資料數量大時，所花時間
bst>array>=dlist



3. Delete

資料數量	時間(s)		
	dlist	array	bst
10	0	0	0
100	0	0	0
1000	0	0	0
10000	0	0	0.01
50000	0	0.03	0.06
100000	0.04	0.01	0.16
500000	0.13	0.19	1.03
1000000	0.23	0.33	2.74

->資料少時差異不大，但資料越多 bst 較其他兩者慢很多



4. 在 1000000 個資料中尋找第 561313 個資料

-> array : 0 秒

-> dlist : 0.03 秒

-> bst : 0.06 秒