Abstract Data Type comparison

B04901066 電機二/洪國嘵

一、資料結構的實做

1. 差異

	Array	Dlist	BST
分布	連續	不連續	不連續
隨機存取	可 (O(1))	不可 (O(n))	不可 (約 O(logn))
插入資料	破壞排序	破壞排序	維持排序(約 O(logn))
刪除資料	破壞排序	維持排序	維持排序
效能較佳的部分	插入、刪除、存取 (O(1))	[插入、刪除、存取](當已取得位置或很靠近頭或尾)	插入、尋找(約 O(logn))
效能較差的部分	排序(O(n^2)或 O(nlogn))	排序(O(n^2)或 O(nlogn))	刪除、隨機存取、iterator++或--

2. 實作方式

Array：和 new 一般的 array 相同，只是當空間不足時，重新 new 兩倍大小的空間，然後把資料搬過去新的空間，再繼續填新的資料，delete 舊的記憶體。

Dlist：資料之間以雙向之指標相連，操作僅能透過第一個 node，並且最尾端為 dummy node 以方便做資料識別和處理。

使用 insertion sort，將第 k 筆資料和 k-1、k-2…比較，直到適當位置再插入(以指標實作)。

BST：node 僅存儲資料和左右 children 的指標，僅能透過 root 去存取所有 node；iterator 儲存一個指標的 stack 來記錄 trace，會紀錄 root 指標、一路上遇到的指標和 node 的指標；特別的是，iterator end 裡面的 stack 是 Tree 裡面最末(最右) node 的所有指標再加上一個 NULL，因此設計--end()就會 pop 掉 NULL，變成最末 node。為了使 Tree 為空時 end==begin，即使 Tree 為空 begin 內還是儲存一個 NULL。

刪除是此種資料結構較麻煩的部分，我一律將 handle 轉成 iterator 來處理，刪除時有三種情況：

無 child：直接刪除並將 parent 內的指標改為 NULL

一個 child：直接刪除並將 parent 內的指標改為自己的 child

兩個 children：以自己++的那個 node 取代掉自己，若那個 node 還有 right child(必定沒有 left child)記得要先 maintain 好(即把其 right child 往上接)。

3. 實作優缺點

Array：大致和一般 array 相同。

Dlist：即便記憶體很分散也能將資料 handle 得很好。insertion sort 最多需要

n^2 次的比較，但是實際動到位址只需要 n 次。
BST：比較好寫，因為若沒有 trace，++和—將十分麻煩，可能在 node 裡面就要再多存 parent，造成 maintain 不易且記憶體用量增加。思考的過程注重使—end 變成最末 node，因次便想出了用 stack.top 是 NULL 來標記 end 的方式。

二、實驗比較

1. 實驗設計

實驗一(O3)：隨機插入 10000 筆資料 -> 排序 -> 隨機刪除 1000 筆資料 -> 刪除前 1000 筆資料 -> 刪除後 1000 筆資料
實驗二(O3)：隨機插入 1000000 筆資料 -> 隨機刪除 1000 筆資料 -> 刪除前 1000 筆資料 -> 刪除後 1000 筆資料
實驗三：以 valgrind 執行各個功能(未開 O3)

2. 實驗預期

Array 時間差不多，Dlist 排序較花時間，BST 刪除較花時間，BST 和 Dlist 記憶體使用量大致和 node 數量呈正比(多兩個指標)，Array 因預留 capacity 故記憶體用量可能比實際資料較大。

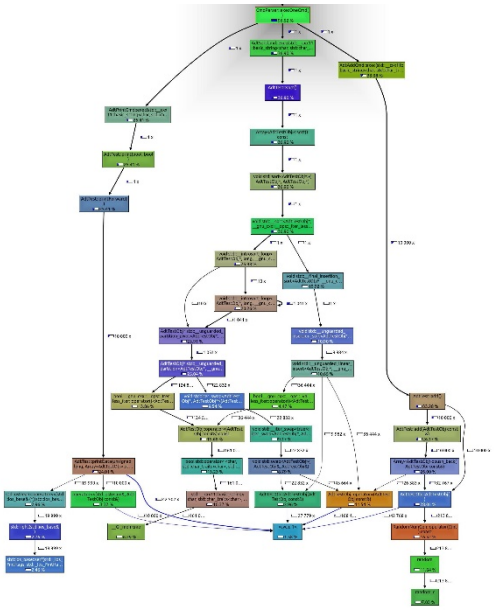
3. 結果比較與討論

實驗一	Array	Dlist	BST
adta -r 10000	0.7422 M	0.4883 M	0.4492 M
adts	0 s	0.26 s	無
adtd -rand 1000	0 s	0.12 s	0.11 s
adtd -front 1000	0 s	0 s	0 s
adtd -back 1000	0 s	0 s	0 s

實驗二	Array	Dlist	BST
adta -r 1000000	48.36 M 0.21 s	61.02 M 0.09 s	61.11 M 0.98 s
adtd -rand 1000	0 s	7.36 s	36.42 s
adtd -front 1000	0 s	0 s	0 s
adtd -back 1000	0 s	0 s	0 s

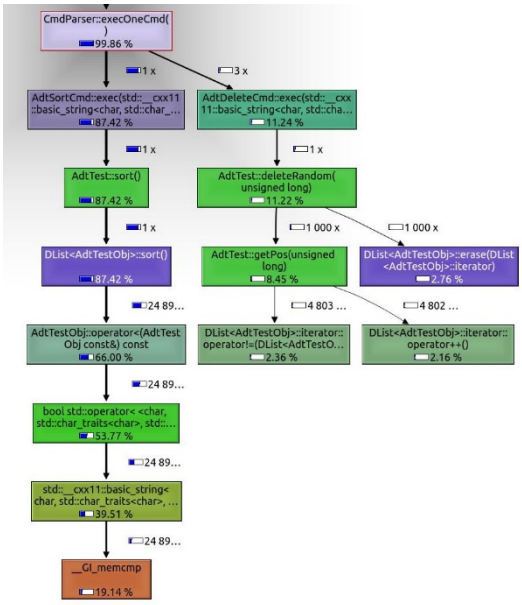
記憶體用量：Array 主要浪費在 capacity，Dlist 和 BST 主要浪費在指標
running time：
adtd –rand 顯示隨機存取的能力，若是沒有要維持資料排序，使用 Array 或 Dlist 似乎較好，若沒有經常的要刪除或++ --，使用 BST 較佳，而且 BST find 的速度亦快許多(約 log n)。

使用 valgrind + kcachegrind 產生以下分析圖：



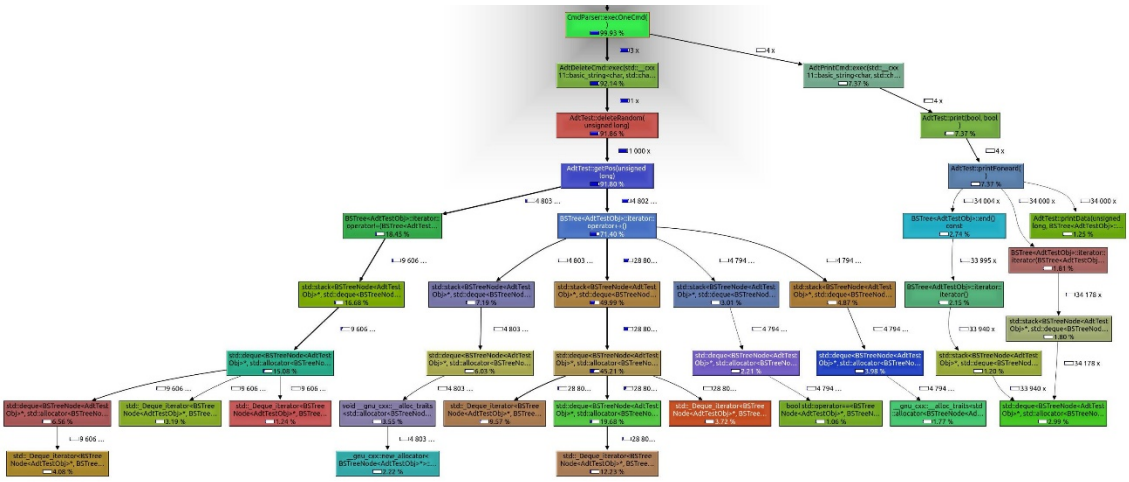
Array

每個 function 使用資源差不多



Dlist

排序使用資源明顯較高



BST

++iterator 使用了大量資源

總結：

Array 的效能無庸置疑應該是最好的，他的缺點是記憶體必須連續，而且動態配置能力不佳，Dlist 十分方便使用，但是因為無法隨機存取所以效能會有點慘，BST 比較適合建立好資料庫後用來搜尋，也不適合用來隨機存取，總而言之，還是得視使用狀況來選擇適合的資料結構。