資料結構與程式設計作業五 Report

電機四 B05901040 蔡松達

（一）實作方法

1. 基本上 array 與 dlist 皆直接依照作業提示的方式進行實作，push_back 將新的 element 擺在最後；當使用 adtdelete 時，不論是從前面、從後面、或 random、全部 pop 掉，都是使用 bool erase(iterator pos)在做：若是從前後 delete，直接 erase(begin())或 erase(--end())即可，全部 delete 掉只需要不停的 pop_front 直到沒有 element 存在為止，若是 random delete 則是直接指定要刪除哪個 element，因此也是使用此 erase function，但若是 delete 特定 string 則是使用 bool erase(const T& x)，必須先用 find()尋找出該 element 的位置（直接求出其 iterator），若 element 存在則直接再帶回 bool erase(iterator pos)去刪除，因此在 bool erase(const T& x)只會進行 find()的動作，實際刪除時還是 call erase(iterator pos)這個 function。

2. BST 實作：每個 node 裡面我只有紀錄左 child 與右 child 的 address，而在整個 BST 中只有紀錄 root 的 address，iterator 中則有紀錄該 node 的 address，以及從 root 到該 node 的路徑(trace：以 stack 實作，一個 element 包含 node address 以及往左或往右走兩個資訊，以 pair 做為 stack 中的 element)。在 BST 為 empty 時 root 將指向 nullptr，此時 begin()與 end()所得到 iterator 中的 node address 都是 nullptr；當 BST 非 empty 時，為了不使用額外的 dummy node 表達出 end()，因此我設計使得 end()的 iterator 之 node address 與最後一個 element(也就是最右邊的 element)相同，但是為了辨識與該 node 是不同的，因此 end()的 trace 會再額外 push 進一個 dummy pair(node address, right)，一般的情況下 stack 的 top 所存的 node address 必定是目前 iterator 所指到的 node 的 parent，但只有在 end()時 top 將會取得與該 node 一樣的 address，換句話說就是 end()表面上取到的是最後一個 element，但其 trace 額外加入了一個 dummy pair 將自己寫入，因此做存取時首先會先判斷 stack 的 top 所存的 node address 是否與本身相同，若相同代表該 iterator 為 end()，若不相同則為一般正常的 node。

3. BST 的操作：insert 時並非直接將數據往最後面擺，而是得先使用 find()，找出要插入的位置點再進行 insert，若出現重複值時則判斷左側是否仍有 child，若仍有則繼續向左側下方尋找插入位置；反之若左邊無 child 則直接插在該處。Delete 則較為複雜，若是要被 delete 的 element 下方沒有或只有一個 child 則直接移除並讓 parent 指向後面的 child 即可；但若是有兩個 child 則必須先找出 successor，讓他補位後再將原先在其下方的資訊正確的接好。其餘操作基本上與 1.相似。

（二）實驗數據

1. Insert n data (1 by 1)：此處以指令"adta -r 1"產生 data，n 為多少便執行多少次該指令基本上在 n 較小時使用 array 的 runtime 與 memory 使用量會是最小的，在 n 越來越大後則較無明顯優勢，而 BST 的 runtime 則較長，主要是因為必須維持整體架構保持 sorted 的狀態。

n=	Dlist		Array		BST	
	Runtime (sec)	Memory (M)	Runtime (sec)	Memory (M)	Runtime (sec)	Memory (M)
1000	0.03	0.043	0.02	0.2422	0.04	0.2578
5000	0.07	0.3438	0.04	0.5938	0.06	0.5
10000	0.12	0.875	0.1	0.8789	0.16	1.012
50000	0.56	4.422	0.68	3.988	0.73	4.746
100000	1.35	9.098	1.19	7.957	1.38	9.18
500000	6.5	45.57	6.53	32.21	8.18	45.86
1000000	13.89	91.38	13.84	64.16	15.39	91.67
5000000	65.67	511.9	66.93	512.1	94.11	512.2
10000000	135.6	1024	141	1024	174.3	1024

2. Insert n data (1 by 1) and destroy the ADT (remove all)：與(一)相同，只在最後加上 "adtd -a"，並且額外觀察 deletion 所花時間

此處主要觀察 deletion 所需的 runtime，可以發現 BST>Dlist>Array≒0，BST 做 deletion 需要一直維持架構為 sorted，執行過程中需要檢查較多細節因此 runtime 最長；Dlist 只需要一直 pop_front 即可，因此不太需要花太多檢查的手續，不過 delete 過程中仍需要一個一個拔除，仍需少許時間完成；Array 則直接將整塊記憶體還回去，無須一步一步刪除，因此 runtime 在 n<10000000 時皆趨近於 0。

n=	Dlist			Array			BST		
	Total Runtime (sec)	Runtime for deletion	Memory (M)	Total Runtime (sec)	Runtime for deletion	Memory	Total Runtime (sec)	Runtime for deletion	Memory (M)
1000	0.03	0	0.043	0.02	0	0.2422	0.04	0	0.2578
5000	0.07	0	0.3438	0.04	0	0.5938	0.06	0	0.5
10000	0.12	0	0.875	0.1	0	0.8789	0.16	0	1.012
50000	0.57	0.01	4.422	0.68	0	3.988	0.74	0.01	4.746
100000	1.35	0	9.098	1.19	0	7.957	1.41	0.03	9.18
500000	6.51	0.01	45.57	6.53	0	32.21	8.36	0.18	45.86
1000000	13.91	0.02	91.38	13.84	0	64.16	15.78	0.39	91.67
5000000	65.79	0.12	511.9	66.93	0	512.1	96.51	2.4	512.2
10000000	135.8	0.22	1024	141	0	1024	179.2	4.89	1024

3. Alternatively insertions and deletions

此處以指令"adta -r 100"產生 data 以及"adtd -r 50"刪除 data 為一組，n 為多少便執行多少組指令

Runtime: BST>Dlist>Array，代表使用 array 進行交錯插入移除時效果較好，且其 memory 使用量也並不會明顯多於其他兩種資料結構；Dlist 雖然搜尋較為複雜（O(n)），但需要的動作幾乎重複且簡單；而 BST 雖然搜尋刪除處較為簡單（O(n lgn)），但因為需要維持整體架構仍為 sorted 故仍須花較多時間。

n=	Dlist		Array		BST	
	Total Runtime (sec)	Memory (M)	Total Runtime (sec)	Memory (M)	Total Runtime (sec)	Memory (M)
100	0.06	0.1367	0.02	0.3242	0.16	0.2656
200	0.23	0.6211	0.03	0.6602	0.69	0.6953
500	1.69	1.402	0.12	1.461	6.75	1.547
800	4.58	2.43	0.2	2.965	17.55	2.555
1000	9.44	2.895	0.29	2.98	33.51	3.055
2000	100.7	5.969	1.17	6.062	209.1	6.082
5000	891.4	15.29	6.35	12.27	1484	15.53

4. Sort the data

此處以指令"adta -r n"產生 n 個 data，接著執行"adts"做 sorting (此處僅紀錄 sorting 所花的時間)

此處針對 sorting 的 runtime 進行觀察：Dlist>Array>BST=0，BST 在新增、移除 element 時為了保持 sorted 狀態耗費不少時間，因而在此無須進行 sort，而 Dlist 需要花上 $O(n^2)$ 的複雜度，導致所需 runtime 相當長；Array 則是使用較快速的 sorting 方法，可讓複雜度處在 O(n lgn)的狀態，且因為記憶體連續較好進行 sort，因而所需時間並不長。

n=	Dlist		Array		BST (always sorted)	
	Sorting Runtime (sec)	Memory (M)	Sorting Runtime (sec)	Memory (M)	Sorting Runtime (sec)	Memory (M)
10000	1.99	0.6953	0.01	0.5703	0	0.7109
20000	9.83	1.145	0.01	1.426	0	1.152
50000	52.21	2.934	0.03	2.895	0	2.992
80000	135.9	4.789	0.04	6.02	0	4.797
100000	213.1	6	0.05	6.074	0	5.945
200000	862.5	11.93	0.12	12.09	0	12.36

整體而言使用 Array 在各方面表現都不錯，尤其是在要反覆新增或移除 element 的情況下最為適合，BST 則較為適合用在需要保持 sorted 狀態的時候，Dlist 則似乎較無明顯優勢。