

ADT Performance study report

電機二 孟妍 b06901066

一、資料結構的實做

1. Array

Dynamic Array 的記憶體是連續的，function 方面基本上就跟 STL 的 static array 一樣，唯一的差別在於記憶體可以被動態配置。在記憶體空間不足時 `push_back()` 就會變得有點麻煩，必須先把 `capacity` 變成兩倍，new 一個新的 `capacity` 大小的 array，把舊的資料 copy 到新的 array 再刪除舊的 array。

2. Dlist

Doubly linked list 的實作就是不斷改變它的 `_next` 跟 `_prev pointer`，因為是雙向鏈接的，所以記憶體可以是不連續的，因此 Dlist 在新增資料與刪除資料只要改變 `pointer` 的指向即可，算是容易的。Dlist 中比較麻煩的部分是 `sort`，我一開始用了最簡單的 `bubble sort`，但發現時間是老師 `ref program` 的好幾倍，因此改用了 `insertion sort`，雖然同樣是 $O(n^2)$ 但平均通常不需要像 `bubble sort` 比較跟移動那麼多次，且在插入新資料時，原資料已排序的狀況下使用 `insertion sort` 較快。

3. BST

(1) BSTNode :

存有 `_left`, `_right`, `_parent` 三個指標標示該 Node 前與左右的 Node。

(2) Iterator class:

在 iterator class 中有四個 function: *Successor*, *Predecessor*, *min*, *max*, 用來輔助 operator ++ 跟 --, 當還未 iterate 到最後一個元素時, ++ 跟 -- 都可以直接呼叫 *Successor* 跟 *Predecessor* 即可。

本來想在 BSTree Class 中使用 *_tail* 紀錄 *end()*, 但發現在插入跟刪除資料時都要不斷更新也點麻煩, 所以就想到在 iterator class 中除了 使用 *_node* 紀錄指到的 current node 外, 另外新增一個一個 *_prev* 的指標用來標記 *end()*, initial 的時候 *_node*, *_prev* 都是 0。當 iterate 到最後一個元素時, 呼叫 ++, 此時 *_node* 的 *Successor* 指標會等於 0, 所以把 *_node* assign 成 0 然後將 *_prev* 設為 *_node*, 因此 *end()* 的判斷就會變為: *if*(*_node*==0 && *_prev*!=0)。當在 *end()* 時呼叫 -- 就可以把 *_node* 設回 *_prev* 的值, 然後將 *_prev* 設回 0。

(3) BSTree

BST 中最麻煩的部分就是刪除元素了, 但因為有 *Successor*, *Predecessor*, *min*, *max* 等函式輔助, 所以有好做一些。大致分為幾種 case:

(a) no child

這種情形最容易處理, 直接刪掉即可

(b) one child

又分為 has left child 跟 has right child 兩種情形, 通常用於呼叫 *pop_front()* 跟 *pop_back()* 時會用到, 這個 case 也是容易處理的, 只需判斷要被刪除的 node

原先接在 parent 的左或右，把它唯一的 child 接到它的 parent 就可以安心地把 node 刪除了。

(c) two children

最麻煩的情形，必須找該 node 的 Successor 取代。大致分為幾種 case:

1) Successor 就是下一個 node (即 Successor 等於 `_node->_right`),

此情形又再分為有無 right child 兩種。

2) Successor 不是下一個 node, (即 Successor 不等於 `_node->_right`),

此情形又再分為有無 right child 兩種。

不管是哪種情形，做法都是先找到 Successor，把 node data 改成 Successor

data，再各種把指標接好，就可以安心刪除了。

二、實驗比較

1. 實驗設計

為了比較綜合 performance，因此執行這新增、排序、刪除、查找三個步驟，

然後觀看 usage:

```
adta -r 50000
```

```
adts
```

```
adtd -r 10000
```

2. 實驗預測

Array 的各項表現中等，但因為記憶體不夠時會 new 兩倍記憶體，所以記憶體用量會較大。Dlist 的 sort 會較慢，因為需要移動 pointer，BST 在 insert 跟 erase 會較慢，因為需要維持排序。

3. 實驗結果與比較

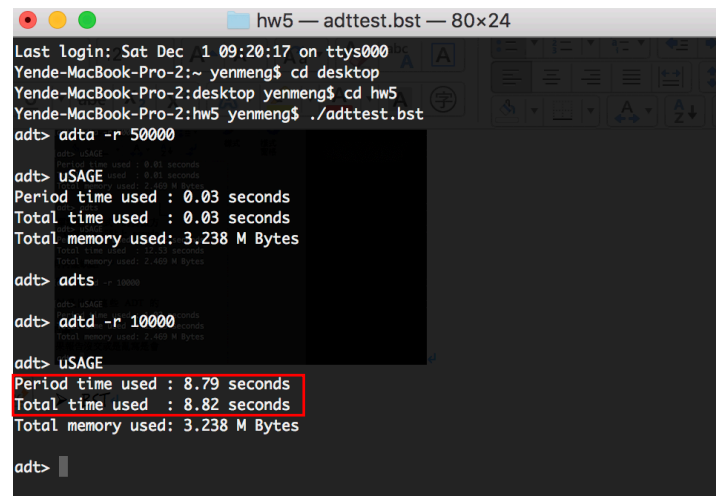
Array

```
hw5 — adttest.array — 80x24
Yende-MacBook-Pro-2:desktop yenmeng$ cd hw5
Yende-MacBook-Pro-2:hw5 yenmeng$ ./adttest.array
[adt> adta -r 50000] formance, 因此執行這新增、排序、刪除三個步驟，然後觀
[adt> uSAGE
Period time used : 0.01 seconds
Total time used : 0.01 seconds
Total memory used: 4.156 M Bytes
[adt> adts
[adt> uSAGE
Period time used : 0.03 seconds
Total time used : 0.04 seconds
Total memory used: 4.156 M Bytes
[adt> adtd -r 10000
[adt> uSAGE
Period time used : 0.01 seconds
Total time used : 0.05 seconds
Total memory used: 4.168 M Bytes
adt>
```

Dlist

```
hw5 — adttest.dlist — 80x24
Yende-MacBook-Pro-2:desktop yenmeng$ cd hw5
Yende-MacBook-Pro-2:hw5 yenmeng$ ./adttest.dlist
adt> adta -r 50000
adt> uSAGE
Period time used : 0.01 seconds
Total time used : 0.01 seconds
Total memory used: 2.469 M Bytes
adt> adts
adt> uSAGE
Period time used : 12.52 seconds
Total time used : 12.53 seconds
Total memory used: 2.469 M Bytes
adt> adtd -r 10000
adt> uSAGE
Period time used : 1.02 seconds
Total time used : 13.55 seconds
Total memory used: 2.469 M Bytes
adt>
```

BST



```
hw5 — adttest.bst — 80x24
Last login: Sat Dec 1 09:20:17 on ttys000
Yende-MacBook-Pro-2:~ ynmeng$ cd desktop
Yende-MacBook-Pro-2:desktop ynmeng$ cd hw5
Yende-MacBook-Pro-2:hw5 ynmeng$ ./adttest.bst
adt> adta -r 50000

adt> uSAGE
Period time used : 0.03 seconds
Total time used : 0.03 seconds
Total memory used: 3.238 M Bytes

adt> adts

adt> adtd -r 10000

adt> uSAGE
Period time used : 8.79 seconds
Total time used : 8.82 seconds
Total memory used: 3.238 M Bytes

adt>
```

➤ 結果比較

跟預測中的差不多，三種 A D T 各有優缺點，像 BST 因為一直是排序好的狀態，所以方便查找，但相對如果要一直新增與刪除資料較不適用。Array 的話因為記憶體是連續的，所以 access 容易，各方面 runtime 都很快，因為不需要像 Dlist 或 BST 一直移動 pointer，唯一確點就是較浪費記憶體。Dlist 的話比較適用於不需要排序的資料，因為 Dlist 新增與刪除資料容易，但排序卻很耗時。