DSnP HW5 Report

b02901013 鄧傑方

一、資料結構的實做

＊ doubly linked list

每個node都存有前面、後面node的指標，所以可以方便的往前往後探索，而其中的關鍵點在於我們有一個dummy node，將這一整個連結串成一個圈，呼叫end()時就回傳這個dummy node，因此也不會印出dummy node的內容，在實現上更為方便。

其中比較特別的地方在於，我原本sort是使用bubble sort，但當有很大一筆內容要sort時，執行起來花費的時間就極為可觀，因此我將其改寫成select sort，每次比較時，把最小的暫存起來，最後只要進行一次交換，比起用bubble sort一直進行交換快了許多。在實測五萬筆資料時，使用select sort費時74.44s，bubble sort費時232.3s，可見真的快了許多。

＊ dynamic array

最初new了一個size為0的array，直接把\_data指向一串連續的資料，當insert判斷到capacity空間不夠的時候，就會new一個新的size為capacity\*2的array，然後把原本的資料複製過去，並把原本的delete掉，雖然在小筆資料時要搬來搬去，但是大筆資料時就比較不會有這種狀況了，因此我覺得一開始可以從64左右開始做，兼顧效率與記憶體的使用。當刪除資料時，我們還是會保存原來的capacity大小，並不會將它縮減。

每當任何的資料erase完後，都要將其之後的資料往前移動，因為我是用\_size這個變數來處理，所以並沒有將最後一項清乾淨，但是因為有\_size，所以就不會讀到這些資料。erase(const T&x)的地方，我利用寫好的erase(iterator pos)來處理，可以減少code長度及bug可能性。

＊ binary search tree

每個node都有\_left、\_right、\_parent，因為沒有parent在實作上感覺比較麻煩，所以我就設了。原本沒有設\_tail，走到end是判斷其\_left、\_right是否為NULL，但這樣會有個問題，就是當print reverse時，回傳的end()是NULL，因此無法進行operator--的動作，想不到其他方式的狀況下只能加了一個dummy node。不過它的位置非常關鍵，若是加在最大值的\_right時，每當erase的是最大值時都有搬動上的考量，因此決定弄一個\_max在\_root的右上方，每當insert時都先判斷，維持\_max為最大值的狀況，呼叫end()時就回傳\_max，這樣在寫起來不會複雜太多，也能執行operator--的動作，

此外也設了\_size這個變數來處理許多增減的狀況，在判斷上比較直覺，就不用透過iterator的方式來得知資料的數量。在erase(iterator pos)的部分，真的很複雜，要考慮的狀況超多，因此我主要將它分成四大類，\_left、\_right皆是NULL的狀況，\_left、\_right其中一個為NULL的狀況，­\_left、\_right皆不是NULL的狀況，最後一個狀況時，我是將其\_right之下的最小值拿來做為連結的node。最後還要判斷刪除的node是否為\_root，考慮的狀況真的不少。erase(const T&x)的部分，一樣是利用寫好的erase(iterator pos)來處理。

二、實驗比較,請說明你所設計的實驗以及比較這些 ADT 的 Performance

1. 隨機的50000筆資料

1.測資：

adta –r 50000

usage

adtd –r 1000

usage

adts

usage

adtd –r 1000

usage

q –f

2.實驗結果預期：

在adta時，因為dlist、array是直接增加資料，bst則有進行排序動作，因此預期bst最慢。adtd –r時，array每次都需要搬動不少資料，bst也是要考慮許多狀況，相較起來dlist只需進行一些連結，所以dlist最快。bst因為已經sort好，所以最快，dlist、array應該差不多。分類後的adtd –r應該還是相同的結果。

3.結果比較討論：

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dlist | Array | BST |
| adta –r 50000 | 0.03/0.02 | 0.03/0.02 | 0.07/0.05 |
| adtd –r 1000 | 0.52/0.23 | 1.29/1.79 | 2.28/3.08 |
| adts | 66.88/60.47 | 0.06/0.05 | 0/0 |
| adtd –r 1000 | 0.22/0.25 | 3.02/2.87 | 3.74/2.97 |

大致上都符合預期，比較讓我驚訝的是array與dlist的sort比較，array竟然只花了0.05左右的時間，看來dlist都把時間花在重新連結上了，而array只是單純換內容，速度當然快上許多。

1. 不平衡的50000筆資料

1.測資：

adta –s 50000

adta –s 49999

.

.

.

adta –s 2

adta –s 1

usage

adtd –s 50000

usage

adtd –s 25000

usage

adtd –s 1

usage

adts

usage

adtd –s 49999

usage

adtd –s 24999

usage

adtd –s 2

usage

q –f

2.實驗結果預期：

在adta -s時，因為dlist、array是直接增加資料，bst則有進行排序動作，因此預期bst最慢，而且因為資料並不平均，預期會比上面的實驗還久。adtd –s時，則會因為刪除資料位置不同而花費時間不同，對bst影響最嚴重。bst因為已經sort好，所以最快，dlist要處理連結的問題，速度會最慢。分類後的adtd –s，三者速度應該差不多。

3.結果比較討論：

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dlist | Array | BST |
| adta –s 50000… | 0.44/0.36 | 0.4/0.39 | 30.9/27.3 |
| adtd –s 50000 | 0/0 | 0.01/0 | 0/0 |
| adtd –s 25000 | 0.01/0 | 0/0.01 | 0/0.01 |
| adtd –s 1 | 0/0 | 0.01/0 | 0.01/0 |
| adts | 61.5/61.17 | 0.1/0.04 | 0/0 |
| adtd –s 49999 | 0/0 | 0/0 | 0/0 |
| adtd –s 24999 | 0.01/0 | 0/0 | 0.01/0 |
| adtd –s 2 | 0/0 | 0/0 | 0/0.01 |

結果大致上也符合預期，不過本來期待在adtd –s 50000時，看到bst會花費較久時間，結果可能是砍的資料太少而看不到明顯的效果。而在adta –s 50000…時所花的時間比上個實驗adta –r 50000來的要久，我想最主要的原因在於他要一直印出adta –s XXXXX吧！