資結HW5報告

Array, DList, BST

B07901010

范詠為

1. 實作部分
2. array.h
3. class iterator

這部分因為array是連續的記憶體空間，所以overloading++/--時基本起只要\_node++/--即可。

1. begin(), end(), empty(), size()

這部分也簡單，begin()/end()分別回傳含有\_data/\_data+\_size的iterator，empty()/size()都可用data member \_size完成。

1. push\_back(const T& x)

如果\_size ==\_capacity，就呼叫expand()去要記憶體空間，並++\_size和把\_isSorted設回false。

Helper Function：

void expand() //要記憶體空間

分成\_capacity是0和不是0的情況討論，前者要一個記憶體空間後\_capacity設成一，後者\_capacity乘二，並將之前的資料複製過去再將原本要的空間歸還給系統。

1. pop\_front(), pop\_back(), clear()

pop\_back()和clear()基本上就是將陣列大小(\_size)減一和設為零，只要不要取到資料就好，後面的資料在push\_back後會被蓋掉。但pop\_front()在\_size>=2時不能這麼做，要將最後一筆資料換到第一個（不在意資料順序的做法）後才能--\_size，在這個情況下因為有動到資料順序，\_isSorted設回false。

1. erase(iterator pos)

跟pop\_front()類似，在\_size>=2時，將最後一筆資料換到pos位置再--\_size，\_isSorted設回false。

1. erase(const T& x), find(const T& x)

前者只要find不是回傳end()，就呼叫erase(iterator pos)，pos為find回傳的iterator。至於find()就是將0~\_size – 1跑一遍看有沒有跟x相等，沒有找到的話就回傳end()。

1. sort()

為內建的sort()函式，有增加判斷\_isSorted為true時就不sort()直接return和sort()完後將\_isSorted設為true兩行。

1. dlist.h

基本上DList的結構重點就只有三個：

\_head -> \_prev is dummy

dummy -> \_next is \_head

dummy -> \_prev is the last element

前兩個任何情況（list不管是不是空的）都必成立。

* 1. DList constructor

一開始\_head是dummy，故\_head -> \_prev、\_head -> \_next都等於\_head。

* 1. class iterator

overloading++/--時是將\_node指到下/上一個，也就是\_node = \_node -> \_next/\_prev。

* 1. begin(), end(), empty()

因為符合上述的三個原則，begin()為\_head，end()為dummy即是\_head -> \_prev，節省走一遍資料的時間。empty()也只要判斷\_head -> \_prev是不是等於\_head就可，不用記住\_size。

* 1. size()

既然沒有記住\_size，勢必得從begin到end前一個跑過一遍來計算。

* 1. push\_back(const T& x)

先new一個DListNode<T>\*，它的\_prev是dummy的\_prev，\_next是dummy。然後我分成兩種討論，如果list是空的（增加第一個元素），那麼\_head、dummy的\_prev、dummy的\_next都要指到它。如果不是空的，那麼要指到它的有dummy的\_prev -> \_next和dummy的\_prev。最後將\_isSorted設為false。接下來的delete功能都不用改變順序。

* 1. pop\_front(), pop\_back(), clear()

前兩個就是把\_node的連結拆掉再重接，pop\_front()要注意\_head的位置會變，跟array不一樣，最後要把\_node delete。clear()從begin到end前一個跑一遍pop\_front()即可。

* 1. erase(iterator pos)

把\_node的連結拆掉再重接即可。

* 1. erase(const T& x), find(const T& x)

前者只要find不是回傳end()，就呼叫erase(iterator pos)，pos為find回傳的iterator。至於find()就是從begin到end前一個跑一遍看有沒有跟x相等，沒有找到的話就回傳end()。

* 1. sort()

一樣先判斷是否\_isSorted來決定要不要sort()，並且sort()完後將\_isSorted設為true。sort()本身我是用bubble sort。

Helper Function：

void swapData(DListNode<T>\* &a, DListNode<T>\* &b) const () //交換data

1. bst.h

BSTreeNodedata member：

T \_data;

BSTreeNode<T>\* \_left;

BSTreeNode<T>\* \_right;

BSTreeNode<T>\* \_parent;

BSTree data member：

BSTreeNode<T>\* \_root;

size\_t \_size;

基本規則如下：

\_root -> \_parent is dummy

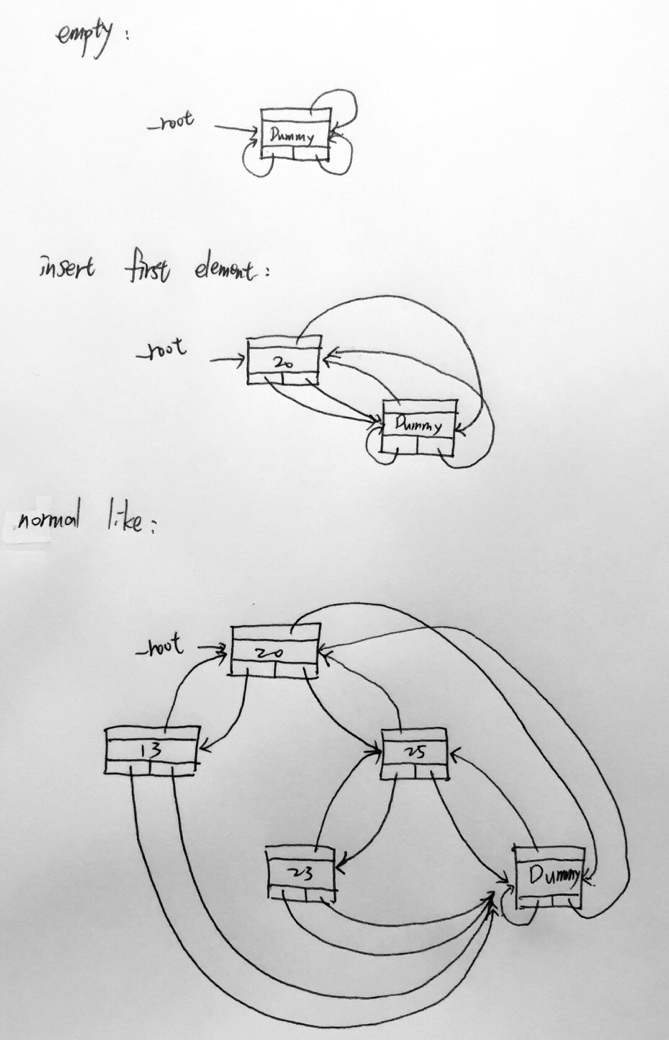
dummy -> \_left is dummy

dummy -> \_right is \_root

dummy -> \_parent is biggest element

leaf -> \_left & \_right are both dummy

架構圖大概長這樣：



但是寫到最後其實覺得沒必要這麼複雜，有試著寫另一版本但因時間來不及而作罷。

1. BSTree constructor

先new一個BSTreeNode即為dummy node亦是\_root、\_root -> \_left、\_root -> \_right、\_root -> \_parent共同所指到的地方，參見上圖empty時的樣子。

1. class iterator

++/--是三種資料結構中最為複雜的，因為意思分別是指到下一個大小的\_node，我的做法是舉++為例，大概是先判有沒有rightchild，有的話就忘rightchild過去再往leftchild一直走到底（dummy前）即為下一個大小的\_node，但如果沒有rightchild，則必須往上走直到自己是別人的leftchild才停下。--的做法就相反而已。那因為要判許多東西所以我在class iterator的private也加了幾個inline function，事後也覺得有點冗。

Helper Function：

inline bool isDum() //是不是dummy

inline bool isLDum() //\_left是不是dummy

inline bool isRDum() //\_right是不是dummy

inline bool isPDum() //\_parent是不是dummy

inline bool isLeaf() // isLDum()&&isRDum

inline bool isLChild() //是不是別人的\_left

inline bool isRChild() //是不是別人的\_right

1. begin(), end(), empty(), size()

begin()就是從\_root一直往左走直到dummy的前一個。end()就是回傳dummy也就是iterator(\_root -> \_parent)。empty()則是檢查\_root -> \_parent是不是\_root。size()因為我有記起來所以就直接回傳\_size。

1. insert(const T& x)

三種情況：

1. empty：\_root、\_root -> \_parent、\_root -> \_right指到它，它自己的三個接點都指到dummy(Helper Function：toDum(BSTreeNode<T>\* & n))。

2. insert的東西最大：跟下一種不一樣的地方在因為dummy -> \_parent要指到最大的node，在insert最大的東西時要把dummy -> \_parent換上去。

3. 其他：insert(\_root, x)。

Helper Function：

void toDum(BSTreeNode<T>\* & n) // n = \_root -> \_parent //dummy

void insert(BSTreeNode<T>\* node, const T& x) // function overloading

這個insert的做法是用recursive，也就是判斷x跟node -> \_data的大小往左或往右再呼叫insert，直到到dummy前，就new一個\_data為x的node

1. pop\_front(), pop\_back(), clear(), erase(iterator pos)

這些都靠erase(iterator pos)來達成，而erase(iterator pos)又是去判斷leaf case或onechild case或twochild case三種人然後分別呼叫Helper Function中的delLeaf(pos)、delOneChild(pos)、delTwoChild(pos)。

Helper Function：

void delLeaf(iterator pos) //\_left&&\_right都是dummy時

void delOneChild(iterator pos) //\_left&&\_right其中之一是dummy

void delTwoChild(iterator pos) //\_left&&\_right都不是dummy，交換successor （twochild case所以一定有），變成onechild case 或leaf case（呼叫前二者）。

1. erase(const T& x), find(const T& x)

前者也是靠find後erase(iterator pos)完成，find的做法是recursive，寫在private Helper Function。

Helper Function：

iterator find(const T& x, iterator& pos) //function overloading

recursive作法，從\_root開始，如果x == \*pos就回傳iterator，不然就依大小find左右下去直到dummy就回傳end()。

1. 實驗部分

針對不同指令做執行時間上的比較如下：

1. ADTSort

BST因為add完就已經sort好，故採用add的時間。

* 1. 10000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.01 | 1.06 | 0 |

* 1. 30000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.01 | 9.66 | 0.01 |

* 1. 50000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.02 | 28.09 | 0.02 |

* 1. 100000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.04 | 108.5 | 0.04 |

* 1. 200000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.07 | 436.1 | 0.1 |

從上面五個的結果和每種資料結構本身的運作方式可推知：

Array應為O(nlogn)（網路上寫的(std::sort())）

DList應為Ｏ(n^2)（bubble sort）

BST應為O(nlogn)（平均情況下, worst case insert n次應為n^2）

而array和dlist用isSorted可大幅減少沒改動到資料順序情況下，第二次sort的時間。

1. ADTDelete
2. -a
3. 50000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0 | 0 | 0.02 |

1. 100000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0 | 0.01 | 0.03 |

1. 1000000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0 | 0.07 | 0.44 |

從上面三個的結果和每種資料結構本身的運作方式可推知：

Array應為O(1)（\_size設為0）

DList應為Ｏ(n)（跑一遍pop\_front()）

BST應為O(n)（跑一遍pop\_front()）

1. -f , -b
2. 50000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| -f秒數(s) | 0 | 0.01 | 0.02 |
| -b秒數(s) | 0 | 0 | 0.02 |

1. 100000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| -f秒數(s) | 0 | 0.02 | 0.04 |
| -b秒數(s) | 0 | 0.02 | 0.02 |

1. 1000000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| -f秒數(s) | 0.01 | 0.07 | 0.43 |
| -b秒數(s) | 0 | 0.08 | 0.39 |

從上面三個的結果和每種資料結構本身的運作方式可推知：

Array應為O(n)（pop O(1) n次每次\_size--或交換再\_size--(不管順序)所以幾乎是0）

DList應為Ｏ(n)（pop Ｏ(1) n次）

BST應為O(nlogn)（pop Ｏ(logn)(找begin) n次）

1. -r
2. 30000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0 | 0.79 | 4.01 |

1. 50000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0 | 2.58 | 11.66 |

1. 100000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.01 | 9.66 | 56.68 |

從上面三個的結果和每種資料結構本身的運作方式可推知：

Array應為O(n)（erase(pos) Ｏ(1) n次還是很快）

DList應為Ｏ(n^2)（erase(pos) Ｏ(1) n次，但每次iterator要移過去O(n)）

BST應為O(nlogn)（erase(pos) O(1) n次，但每次iterator要移過去O(n)，但平均是logn）

1. -s （用自己寫的gendo file去跑多筆資料的結果，不然一次只能手動刪一個）
2. 200000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 9.1 | 18.78 | 4.44 |

從上面的結果和每種資料結構本身的運作方式可推知：

Array應為O(n^2 + n)=O(n^2) （find為O(n)，find n次，erase(pos)亦需O(n)）

DList應為Ｏ(n^2 + n^2) )=O(n^2)（find為O(n)，find n次，erase(pos)亦需O(n^2)）

BST快很多應為O(nlogn + nlogn)=O(nlogn)（find為O(logn)，find n次，erase(pos)亦需O(nlogn)）

這個結果跟find有關故不測ADTQuery。

1. ADTAdd
2. 1000000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.13 | 0.09 | 0.98 |

1. 5000000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 0.84 | 0.45 | 7.81 |

1. 10000000筆資料

|  |  |  |  |
| --- | --- | --- | --- |
|  | ./adtTest.array | ./adtTest.dlist | ./adtTest.bst |
| 秒數(s) | 1.66 | 0.9 | 18.32 |

從上面三個的結果和每種資料結構本身的運作方式可推知：

Array應為O(n)

DList應為Ｏ(n)

BST應為O(nlogn)（insert為O(n)，但可看成平均logn，insert n次）