HW3 資料結構之效率比較

409410050 王謙靜

壹、 研究目的

比較資料結構 Linked list、Array、Array with Binary Search、Binary Search Tree 及 Hash 的資料建置時間以及查詢時間。

貳、 測試環境

作業系統: Windows 上的 WSL

CPU: Intel(R)Core(TM)i7-7500U CPU @2.70GHz 2.90GHz

RAM:8.00GB

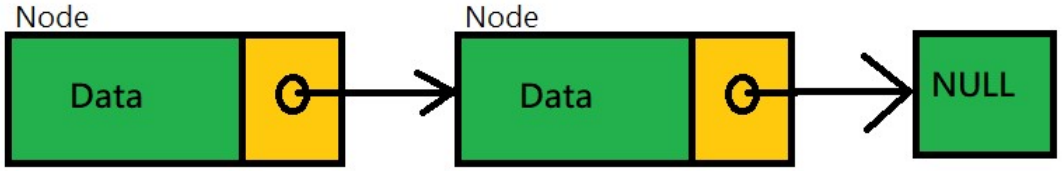
參、 資料結構說明

A. Linked list

i. Linked list 是由許多個點串再一起的資料結構，每個點會存有一些資料(例如整數或是字串)及一個指向下一個節點位置的指標。

ii. Insert: 可以動態將資料插入，放入資料可分為兩種:insert at front 和 insert at tail，從前面放入資料的時間複雜度為 O(1)，從後面插入的資料複雜度是 O(n)（若無特別紀錄尾巴節點的情況)，insert at front 做 traverse 可得到 stack 的效果，insert at tail 則像是 queue。

iii. Find: 從 Linked list 的開頭開始找，一一往後比對，直到找到想找的資料或者到 linked list 的結尾。時間複雜度 O(N)



B. Array

i. 最基礎也最簡單的資料存取結構，以 idx 存取資料。使用時須在 compile time 給定一個固定的值，屬於一個靜態的資料結構

ii. Insert: 若目前陣列存放的數量小於陣列可存放之最大值則將新插入值放入陣列的最後。若陣列滿了，就不能插入了，靜態資料結構的缺點。

iii. Find: 用 idx 從 lowbound 走到 highbound 一一尋找。複雜度 O(N)

以下方為例: lowbound = 0, highbound = 4。

idx	0	1	2	3	4
value	100	25	36	49	3

C. Array with Binary Search

i. 以 Array 為基礎，並在所有資料插入完畢後將其排序，此次實驗使用 c 語言函示庫的 qsort 排序，並在查詢時使用二分搜。

ii. Insert: 和 array 的方法相同，差異則是在所有資料插入後會進行排序。

iii. Find: 由於經過排序，故找資料時可以使用二分搜(binary search)，在二分搜演算法中，每次的可能值區間都會縮小一半，每次查詢都縮小一次可能性值到找到值，或者可能區間長度為 0。

Example: find 5 in array below

	left		mid		right
Idx	0	1	2	3	4
value	3	7	11	29	160

	left	mid	right		
Idx	0	1	2	3	4
value	3	7	11	29	160

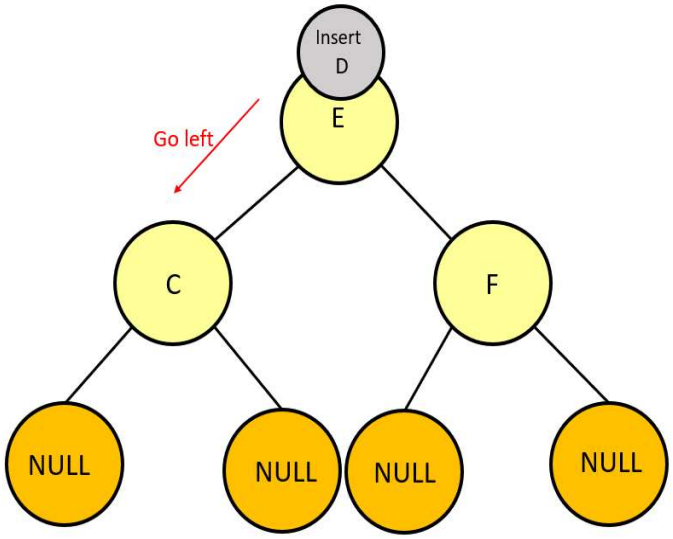
	mid left	right			
Idx	0	1	2	3	4
value	3	7	11	29	160

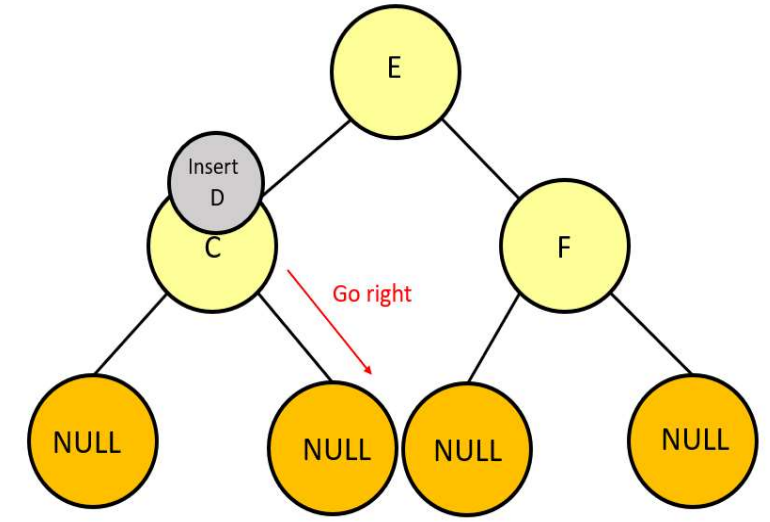
		right left	→ Size = 0 → end		
Idx	0	1	2	3	4
value	3	7	11	29	160

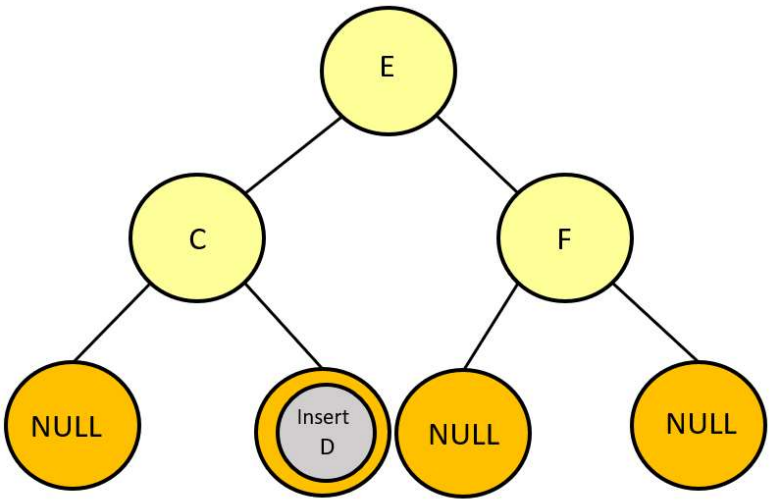
D. Binary Search Tree

i. Binary Search Tree 是一種樹的結構，並保證每個節點最多只有兩個 children，且每個點的左邊的 child 的值小於等於那個點的值，且右邊的 child 大於那個點的值

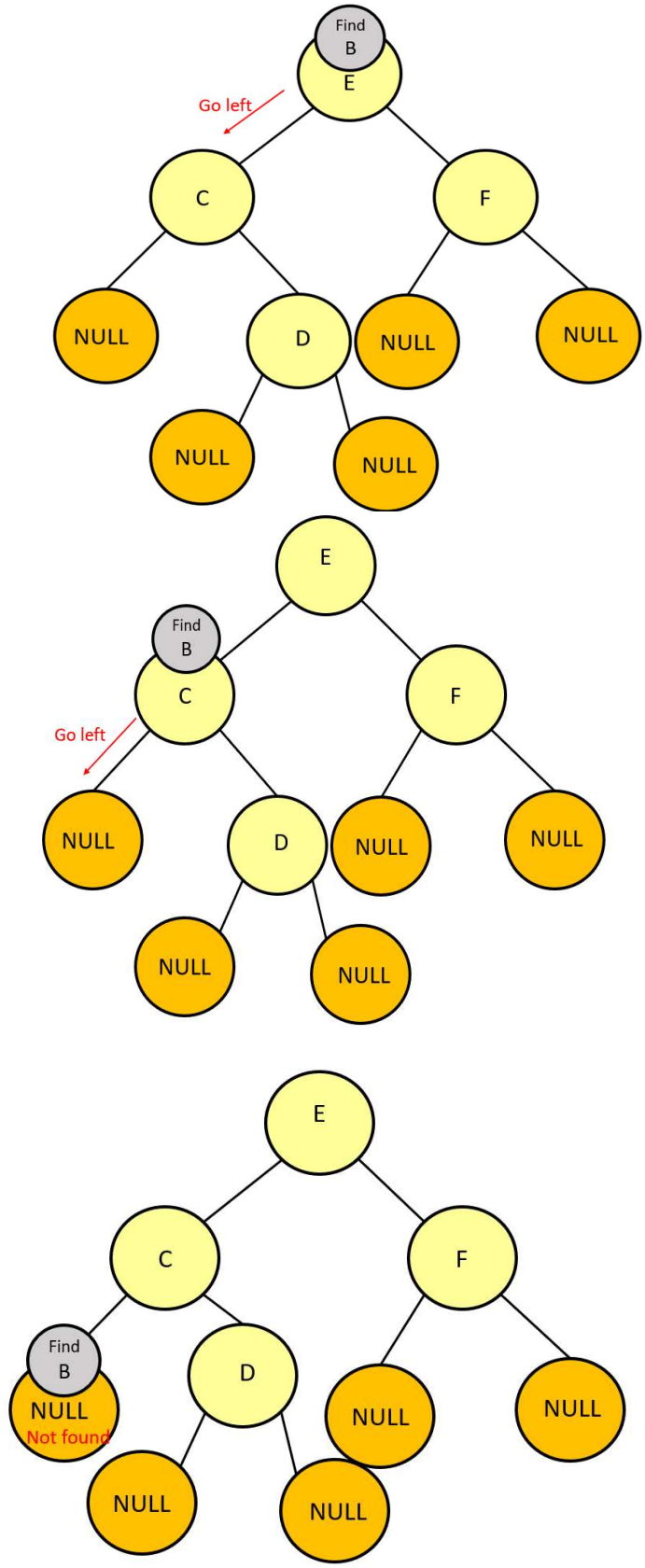
ii. Insert: 從樹根開時走，大於樹根往右走，其他往左走，走到 NULL，將點插入。





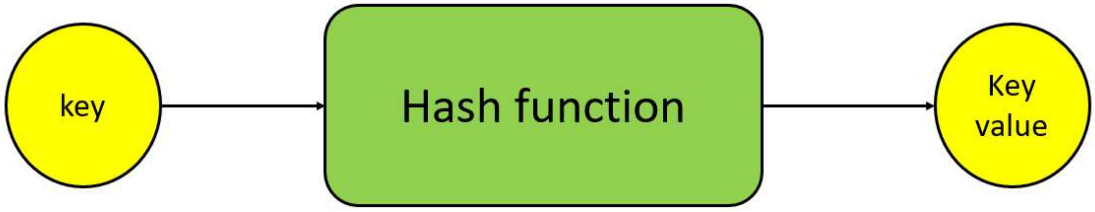


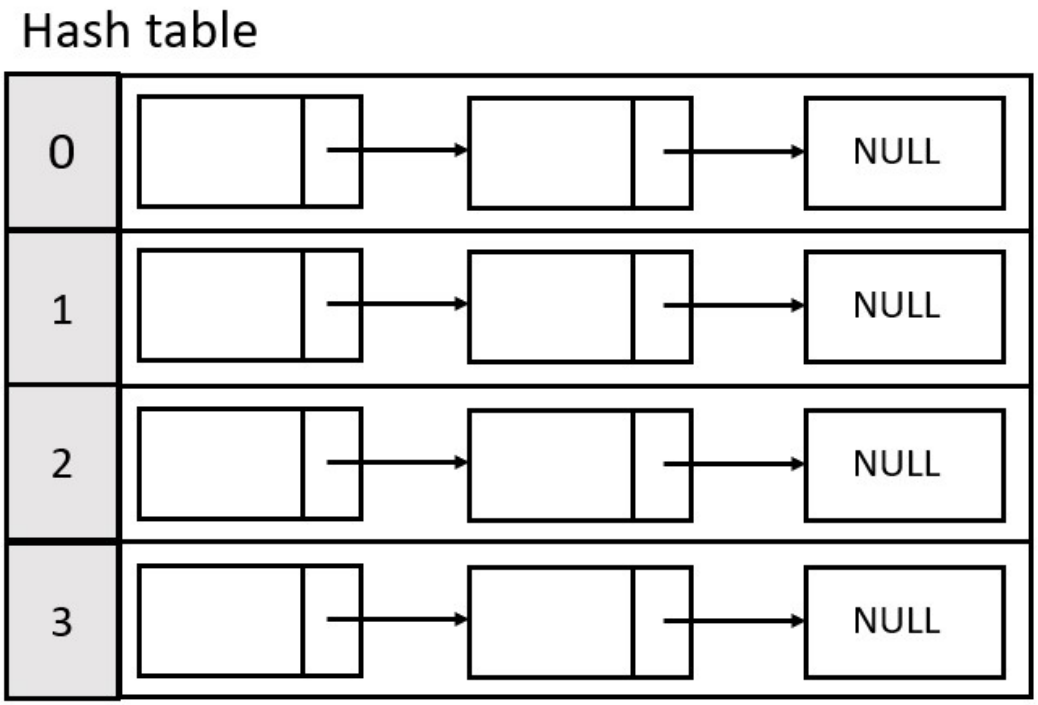
iii. Find: 從樹根開時走，大於樹根往右走，小於往左走，走到找到相同的值或 NULL 結束。


Find
B
E
Go left
C
F
NULL
D
NULL
NULL
NULL
NULL
E
Find
B
C
F
Go left
NULL
D
NULL
NULL
NULL
NULL
E
C
F
Find
B
D
NULL
NULL
NULL
Not found
NULL
NULL

E. Hash

i. 利用 hash function 產生一個 key-value，再利用 key-value 來將值放進 hash table，或者對 hash table 查找。這個報告中的 hash table 以 Linked list 陣列實作，使碰撞的 key 一起存在同一個 key-value。





ii. Insert: 將 key 經由 hash function 得到 key value，並將其放入 key value 所屬的 Linked list 中。

iii. Find: 同樣將 key 經由 hash function 得到 key value，並在 key value 所屬的 linked list 中尋找 key

肆、 建立 Data 方法及數量

使用 rand()函式並將出來的值對 52 取模，0~25 是大寫的(A~Z)，26~51 是小寫的(a~z)，每找出 8 個字母為一開始的字串。之後 rand()對 8 取模為 x，並生成一個字母 c，更改上一個字串第 x 的值為 c。找到新的字串後一一比對確定與其他字串皆不同時放入 data。

伍、 測量排序時間的方式

使用 gettimeofday 函式測量時間。分別在 build、query 開始前、結束後記錄下系統的時間，將結束減去開始可得到排序所需的時間。

陸、 實驗結果

資料結構		Linked list	Array	binary search	BST	Hash
建立: 1e4	Build time	0.001193 sec	0.000588 sec	0.002126 sec	0.004950 sec	0.002784 sec
查詢: 1e3	Query time	0.062366 sec	0.002176 sec	0.000227 sec	0.000292 sec	0.000229 sec
建立: 1e5	Build time	0.013025 sec	0.007186 sec	0.032756 sec	0.073866 sec	0.038590 sec
查詢: 1e4	Query time	9.122660 sec	4.939341 sec	0.003127 sec	0.006820 sec	0.003214 sec
建立: 1e6	Build time	0.175153 sec	0.139129 sec	0.407121 sec	1.25993 sec	0.369029 sec
查詢: 1e5	Query time	1110.299346 sec	644.199660 sec	0.070565 sec	0.132069 sec	0.028370 sec
建立時間複雜度		O(N)	O(N)	O(NlogN)	O(NlogN)	O(N)
查詢時間複雜度		O(N*N)	O(N*N)	O(NlogN)	O(NlogN)	O(N)
靜態/動態結構		動態	靜態	靜態	動態	動態

附上測試結果:

```
root@LAPTOP-HNTDAHGN:/mnt/d/PD/hw2# ./a.out -d 1e4 -q 1e3 -bst -bs -arr -ll -hash
bst:
building time: 0.004590 sec
query time: 0.000292 sec

binary search:
building time: 0.002126 sec
query time: 0.000227 sec

array:
building time: 0.000588 sec
query time: 0.002167 sec

linked list:
building time: 0.001193 sec
query time: 0.062366 sec

hash:
building time: 0.002784 sec
query time: 0.000229 sec
```

```
root@LAPTOP-HNTDAHGN:/mnt/d/PD/hw2# ./a.out -d 1e5 -q 1e4 -bst -bs -arr -ll -hash
bst:
building time: 0.073866 sec
query time: 0.006820 sec

binary search:
building time: 0.032756 sec
query time: 0.003127 sec

array:
building time: 0.007186 sec
query time: 4.939341 sec

linked list:
building time: 0.013025 sec
query time: 9.122660 sec

hash:
building time: 0.038590 sec
query time: 0.003214 sec
```

```
root@LAPTOP-HNTDAHGN:/mnt/d/PD/hw2# ./a.out -d 1e6 -q 1e5 -bst -bs -arr -ll -hash


bst:
building time: 1.259993 sec
query time: 0.132069 sec

binary search:
building time: 0.407121 sec
query time: 0.070565 sec

array:
building time: 0.139129 sec
query time: 644.199660 sec

linked list:
building time: 0.175153 sec
query time: 1110.299346 sec

hash:
building time: 0.369029 sec
query time: 0.028370 sec
```

柒、 總結

在相同的時間複雜度下，靜態的資料結構比動態的資料結構快上許多，利如:普通陣列對上 linked list、排序好的陣列對上 BST，由此可知動態資料的代價是更大的時間常數。普通陣列和 linked list 的優點在於單次 insert 的時間非常快速 O(1)，但單次查詢時間卻也是所有資料結構中最慢的 O(N)，故在使用於插入量遠大於查詢量時，是一個好的資料結構。BST 在 insert 和 query 間取的了較多的平衡，單次操作複雜度皆為 O(logN)，而且外加有排序的性質，但卻有著退化成 Linked list 的風險。排序好的陣列，在單個插入因為要排序故時間複雜度為 O(N)，但若插入完畢後查詢，可使用一般陣列存完後再排序，如此一次 insert 的時間複雜度為均攤 O(logN)，之後使用二分搜查詢複雜度為 O(logN)。然而插入及查找皆 O(1)的 Hash 則是有相當良好的時間效能，但在空間使用上，因為了維持 Hash 的最佳效能，故必須開兩倍大的記憶體。

資料結構	Linked list	Array	binary search	BST	Hash
優點	動態、插入快	插入最快	具有排序性質	動態、有排序性質	查詢、插入快
缺點	查詢慢	查詢慢	無法動態插入	可能退化	記憶體需求較大

捌、 Github 連結

https://github.com/forward0606/PD_hw2_datastructure

玖、 參考資料

AVL tree:

https://josephjsf2.github.io/data/structure/and/algorithm/2019/06/22/avl-tree.html

https://stackoverflow.com/questions/3955680/how-to-check-if-my-avl-tree-implementation-is-correct