HW3 資料結構之效率比較

408410113 王興彥

本次報告內容討論四種資料結構(Linked list, array, tree, hash)其建立及搜尋演算法的實作，

並比較其差異。

GitHub: https://github.com/xingyan0523/pdhw3

1. 實作方法與優缺點

(1) Linked list

Insert 時先建立新的 node，並將 node 插入到最後一個節點，因為這次側資較大，所以而外建立了一個新的結構叫做 list，會記錄 Linked list 的 head 和 rear，讓插入時只需時間複雜度 O(1)，不過也可以直接插入在 head 一樣會是 O(1)。

Search 的部分因為 insert 時沒有照大小插入，所以無法加快搜尋，搜尋的時間複雜度為 O(n)。

優點:資料儲存的量是動態的，insert 只需 O(1)，若要 insert 到指定位置雖然需要 O(n)但不用搬動大量資料。

缺點:要 search 資料時，無 index 故只能循序存取時間複雜度為 O(n)，且每筆資料都須多花額外的指標空間。

(2) Array

Insert 非常簡單，直接找到最後一項並放入資料即可，因為有紀錄最後一項的位置因此只需 O(1)即可完成。

Search 這邊分成兩種一種是循序搜尋，另一種是先經過排序後做二元搜尋，循序搜尋最差為 O(n)，二元搜尋雖然為 O(logn)但是必須先花額外的時間做排序 O(nlogn)，看似由 O(nlogn)決定整體複雜度，但是當搜尋次數達到一定的量時，反而會更耗時間，因此整以來看多花時間做排序後採取二元搜尋是值得的，後續的實驗結果也將驗證此推測。

優點:實做簡單，有 index 在處理資料時比較方便且快速，使用連續的記憶體 CPU 讀快取時會提升效能。

缺點:若是要將資料插入在特定位置，需要搬動大量資料，且資料量如果一直在變更時需要花時機在 malloc 空間以及搬動舊資料。(這次實作為了省時直接開了一個一樣大的陣列，實際上並不會這樣實作)

(3) Tree (BST)

一開始先建立 node 結構，除了 key 之外還要有 left 和 right child，insert 一筆資料進 BST(binary search tree)時，從 root 開始比較若新的資料大於 root 的資料則往右邊走，反之往左，這樣建立資料讓我們在搜尋時能更便利，這樣子的建立方式最差時間複雜度為 O(n)，看整體來說平均複雜度為 O(h) or O(logn)。

Search 資料時跟 insert 時差不多，要找的資料比 root 大則往右，反之往左，兒時間複雜度則跟 insert 是同樣的情況。

優點:他的資料數量也可以是不固定的，平均而言 BST 的操作也可以維持在 O(logn)，且如果我們要得到排序過的資料，只需要使用 in-order traverse 就可以實現 。

缺點:若今天 BST 不為平衡或是有歪斜的情況，操作得速度不但不會變快，還要浪費指標的記憶體空間，因此為了讓他維持平衡，一般實作上都會是 AVL tree 或是紅黑樹等資料結構。

(4) Hash

Hash 的實作有很多方式，這次報告只討論自己實作的方式，首先建立一個 node 的指標陣列，大小為一個質數，陣列裡面的每一個指標代表著一個 linked list，是後續碰撞時的處理方式。

這邊定義的 hash function 也是最基本的 mod 運算，送入資料後，function 會回傳資料 mod 一開始選擇的質數，而它代表著資料 hash 到的 table index。

Insert 時，將資料丟進 hash function，並將資料放到對應的位置，若是之前已經有資料放入，則會發生碰撞，這邊使用 linked list 來做處理，將新進來的資料放到後面接起來，insert 的時間複雜度為 O(1)。

Search 跟 insert 時相同，將資料丟進 hash function，找到對應位置後開始找，沒有就往下一個 node 找，直到 NULL，search 的時機複雜度比較特別，他會取決於 hash function 的實作，若 hash function 很爛，將所有的東西存到同一個位置，則時間複雜度為 O(n)，但一般來說平均為 O(1+α) 其中 α=n/m「資料數量(n)」與「slot 個數(m)」的比例稱為 load factor，若具有 n=O(m)的關係，再加上比較正常的 hash function，則會接近 O(1+α) = O(1+constant) = O(1)。

2. 測試環境與方法

(1) 環境

```
Operating System: Kubuntu 20.04
KDE Plasma Version: 5.18.5
KDE Frameworks Version: 5.68.0
Qt Version: 5.12.8
Kernel Version: 5.8.0-49-generic
OS Type: 64-bit
Processors: 4 × Intel® Core™ i7-6500U CPU @ 2.50GHz
Memory: 11.6 GiB of RAM
```

(2) 方法

(a) 支援參數

-d N 插入的資料筆數，每筆資料是唯一的 (1e4, 1e5, 1e6)

-q M 查詢資料筆數 (1e3, 1e4, 1e5)

-bst: 測量 BST 建立和查詢所需時間

-bs: 測量 Binary Search 建立資料結構(含 sorting) 和查詢所需時間

-arr: 測量 array 建立資料結構和查詢(linear search)所需時間

-ll: 測量 linked list 建立資料結構和查詢所需時間

-hash 測量 hash 建立資料結構和查詢所需時間

(b) 測資產生

這次測試使用測資為整數型態，使用 rand 產生並存在陣列內當作輸入測資，為確保資料唯一，而外開了一個大小為 RAND_MAX 的 bool 陣列，產生測資後將對印 index 設為 true，若遇到生成測資 index 已經為 true，則再次產生測資。

(c) 使用 gettimeofday 來獲取所需時間

3. 測試結果

time: sec	Linked list	Array	Array(BS)	BST	Hash
Insert	0.046604	0.009160	0.251348	1.491166	0.073714
Search	542.149135	264.170302	0.060305	0.185850	5.753608

4. 複雜度(理論)

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

5. 參考資料

https://www.bigocheatsheet.com/