資料結構比較

408410111 林彥廷

Github 連結:

https://github.com/stanley408410111/C-program.git

一、 測試環境

OS: Ubuntu 20.04.2 LTS

CPU : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz

RAM: 4000704 kB

二、 解釋各種資料結構

1. Array

Array 是存儲在連續內存位置的項目的集合，並將相同類型的多個項目存儲在一起。

查詢方式 :

第一種作法 : sequential search 循序的從頭到尾查詢，可想而知此方法非常花時間。

第二種作法 : binary search (陣列需先排序)

搜尋過程從陣列的中間元素開始，如果中間元素正好是要搜尋的元素，則搜尋過程結束；如果某一特定元素大於或者小於中間元素，則在陣列大

於或小於中間元素的那一半中搜尋，而且跟開始一樣從中間元素開始比較。這種搜尋演算法每一次比較都使搜尋範圍縮小一半。

2. Linked List

Linked List 是使用 node 來儲存資料，並利用每個 node 中的 pointer 指向下一個 node ，藉此將多個 node 串連起來。

3. Binary Search Tree

Binary Search Tree 是基於 binary tree 的資料結構，具有以下特性：

節點的左子樹僅包含小於節點 key 的節點。

節點的右子樹僅包含大於該節點 key 的節點。

左子樹和右子樹也都必須是 binary search tree。

4. Hash

利用 hashing 建立一個 hash table。在 hash table 中資料以陣列形式被存儲，其中，每個資料有其唯一的索引值(index)。至於 hashing function 則有各式樣種類的函數。

三、 建立 Data 方法及數量

接收參數建立對應的資料數量

1e4 代表 10000 筆

1e5 代表 100000 筆

1e6 代表 1000000 筆

說明： 因為有資料排序的需求 (binary search) ，因此先依照 array 的 index 建立對應的數字，再利用類似洗牌的方式打散順序(Fisher-Yates 演算法) ，既省去排序的時間也確保產生唯一的數值。

```
void Insert_data(int a[],int sorted_a[],int num)
{
    srand( (unsigned)time( NULL ) );
    for(int i=0; i<num; i++)
    {
        a[i] = i;
        sorted_a[i] = i;
        //printf("%d\n",a[i]);
    }
    for (int i = num - 1; i > 0; i--)
    {
        // Pick a random index from 0 to i
        int j = rand() % (i+1);

        // Swap arr[i] with the element at random index
        swap(&a[i], &a[j]);
    }
    //printf("Data generated successfully\n");
}
```

Fisher-Yates 演算法說明 ：

1. 取隨機值 rand(index)
2. Swap(shuffled[index] , shuffled[隨機值])

重複第 1 和 2 步直到 index = length -1

四、 測量排序時間方式

方法 ： 利用 gettimeofday()函式測量

```
#include <sys/time.h>
#include <unistd.h>

 struct  timeval start;
 struct  timeval end;
 unsigned  long diff;

gettimeofday(&start,NULL);
```

...資料結構新增/查詢...

```
gettimeofday(&end,NULL);

diff = 1000000*(end.tv_sec-start.tv_sec)+ end.tv_usec-start.tv_usec;
float sec;
sec=(float)diff/1000000;
printf("第%d次所需時間為 %f (秒)\n",n,sec);
```

五、 實驗結果

./a.out -d 1e4 -q 1e4 -bst -bs -arr -ll –hash :

```
number of insert data : 10000
number of query data : 10000

bst:
building time : 0.2518 sec
query time : 0.0111 sec
-------------------------
bs:
building time : 0.0002 sec
query time : 0.0125 sec
-------------------------
arr:
building time : 0.0002 sec
query  time : 0.1181 sec
-------------------------
linked list:
building time : 0.1368 sec
query time : 0.0117 sec
-------------------------
hash:
building time : 0.0301 sec
query time : 0.0432 sec

real    0m0.617s
user    0m0.594s
sys     0m0.020s
```

./a.out -d 1e4 -q 1e5 -bst -bs -arr -ll –hash :

```
number of insert data : 10000
number of query data : 100000

bst:
building time : 0.2497 sec
query time : 0.1146 sec
-------------------------
bs:
building time : 0.0002 sec
query time : 0.1272 sec
-------------------------
arr:
building time : 0.0001 sec
query  time : 1.1890 sec
-------------------------
linked list:
building time : 0.1384 sec
query time : 0.1155 sec
-------------------------
hash:
building time : 0.0271 sec
query time : 0.4003 sec

real    0m2.401s
user    0m2.346s
sys     0m0.050s
```

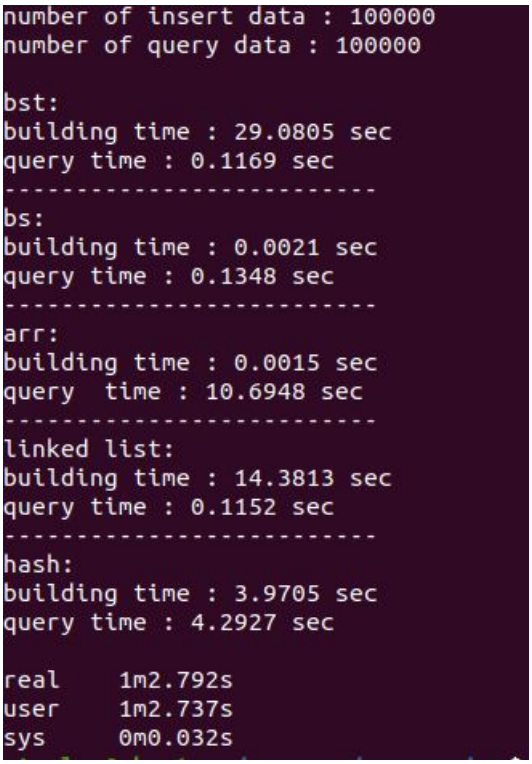
./a.out -d 1e4 -q 1e6 -bst -bs -arr -ll –hash :

```
number of insert data : 10000
number of query data : 1000000

bst:
building time : 0.2632 sec
query time : 1.1546 sec
-------------------------
bs:
building time : 0.0003 sec
query time : 1.3021 sec
-------------------------
arr:
building time : 0.0001 sec
query  time : 11.8257 sec
-------------------------
linked list:
building time : 0.1434 sec
query time : 1.1299 sec
-------------------------
hash:
building time : 0.0284 sec
query time : 3.9488 sec

real    0m19.798s
user    0m19.779s
sys     0m0.012s
```

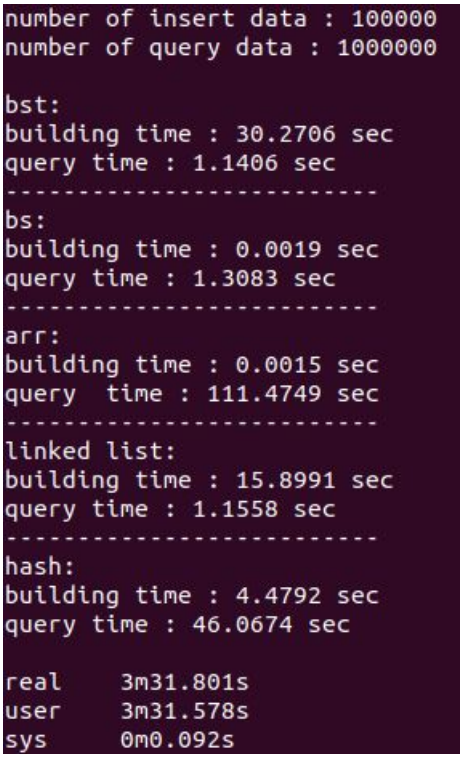
./a.out -d 1e5 -q 1e4 -bst -bs -arr -ll –hash :

```
number of insert data : 100000
number of query data : 10000

bst:
building time : 28.3945 sec
query time : 0.0111 sec
-------------------------
bs:
building time : 0.0442 sec
query time : 0.0138 sec
-------------------------
arr:
building time : 0.0017 sec
query  time : 1.0683 sec
-------------------------
linked list:
building time : 14.6460 sec
query time : 0.0114 sec
-------------------------
hash:
building time : 4.2364 sec
query time : 0.4194 sec

real    0m48.849s
user    0m48.297s
sys     0m0.536s
```

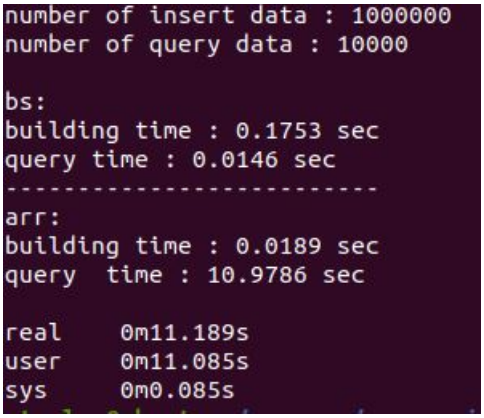
./a.out -d 1e5 -q 1e5 -bst -bs -arr -ll –hash :



./a.out -d 1e5 -q 1e6 -bst -bs -arr -ll –hash :

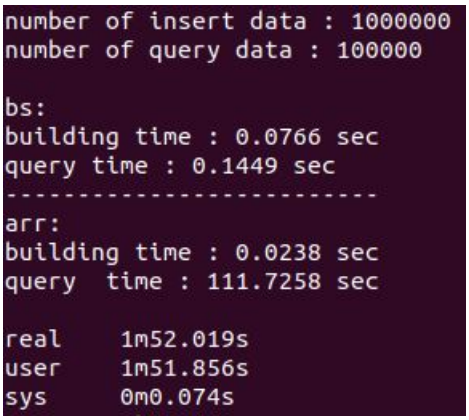


./a.out -d 1e6 -q 1e4 -bst -bs -arr -ll –hash :



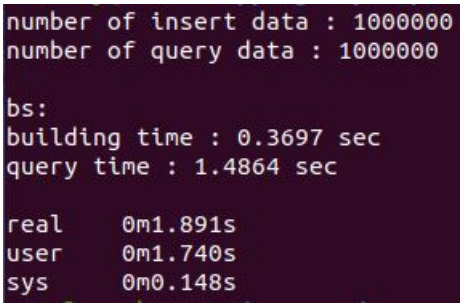
(-bst -ll -hash 時間大於 10 分鐘)

./a.out -d 1e6 -q 1e5 -bst -bs -arr -ll –hash :



(-bst -ll -hash 時間大於 10 分鐘)

./a.out -d 1e6 -q 1e6 -bst -bs -arr -ll –hash :



(-bst -arr -ll -hash 時間大於 10 分鐘)

六、 總結

1. Array

好處: Array 較 linked list 節省記憶體空間，linked list 需要額外的記憶體空間來儲存指到下一個 node 的 pointer。

壞處: 沒有像 linked list 能動態新增和刪除元素的資料結構特性：array 的元素在記憶體中是連續儲存的，而 linked list 的元素為非連續儲存的。

使用時機：

甲、 希望能夠快速存取查詢資料

乙、 已知欲處理的資料數量

丙、 要求記憶體空間的使用越少越好

2. Linked List

好處: 新增和刪除資料較 array 簡單，且 Linked list 的資料數量可以是動態的，不像 array 會有 amortization 的 resize 問題。

壞處: linked list 沒有 index，若要查詢特定 node

的資料，需要從頭開始找，需要額外的記憶體空間來儲存指到下一個 node 的 pointer。

使用時機 ：

甲、 需要頻繁地新增/刪除資料時

3. Binary Search Tree

好處: 高效的插入和刪除，較 sorted array 和 linked list1 插入和刪除速度更快

壞處: 如果我們使用不平衡的 binary search tree，則會導致性能不佳。

4. Hash

好處: 在許多情況下，hash table 比搜索樹或任何其他表搜尋結構更有效率。因此，hash table 被廣泛用於軟體中，尤其是用於數據庫索引。

壞處: hash collision 實際上是不可避免的。當 hashing 大量資料時，可能會發生許多 collision，此時 hash table 的效率變得非常低。

七、 參考資料

1. https://medium.com/@asd757817/%E7%B0%A1%E5%96%AE%E5%8F%88%E8%A4%87%E9%9B%9C%E7%9A%84%E6%B4%97%E7%89%8C%E6%BC%94%E7%AE%97%E6%B3%95-7e7254bb9145
2. https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%88%86%E6%90%9C%E

5%B0%8B%E6%BC%94%E7%AE%97%E6%B3%95
3. https://codertw.com/%E5%89%8D%E7%AB%AF%E9%96%8B%E7%99%BC/196959/
4. https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm
5. https://medium.com/@maggieliao.cm04g/%E8%B3%87%E7%B5%90%E8%88%87%E6%BC%94%E7%AE%97%E6%B3%95%E7%AD%86%E8%A8%98-1-linked-list-%E8%88%87-array-%E6%96%BCo-n-%E4%B9%8B%E5%B7%AE%E7%95%B0%E6%AF%94%E8%BC%83-badbf08b17ce
6. https://en.wikipedia.org/wiki/Hash_table
7. https://www.geeksforgeeks.org/binary-search-tree-data-structure/
8. https://www.datacamp.com/community/tutorials/git-push-pull
9. https://stackoverflow.com/questions/3025050/error-initializer-element-is-not-constant-when-trying-to-initialize-variable-w
10. https://stackoverflow.com/questions/47299592/how-to-initializing-a-hash-table-in-c
11. https://blog.techbridge.cc/2017/01/21/simple-hash-table-intro/
12. https://stackoverflow.com/questions/61697683/insertion-in-hash-table-using-singly-linked-lists
13. https://www.youtube.com/watch?v=V1L2mIZePR4
14. https://practice.geeksforgeeks.org/problems/advantages-and-disadvantages-of-hash-table