資料結構之效率比較

Comparison of data structures

408530014. - 陳品洋 - 2021年5月21日

前言

這次的資料結構測試包含了，陣列、排序後的陣列（二元搜尋用）、鏈結串列、二元樹。由於作業要求每筆資料必須為唯一，在這裡我的作法為建立好資料結構後，在做重新遍歷找出重複的直並重新設一亂數。使用while直到部資料結構裡沒有同樣的值。但是重新遍歷的時間不計算以求公平。總之，在這次作業裡也複習到許多先前學習到的知識，相信穩固好這些觀念，程式設計的能力會大幅增加。



系統與環境

作業系統：macOS Mojave10.14.6
cpu處理器：2.3 GHz i5
記憶體：8 GB 2133 MHz LPDDR3
開發整合環境: vim

資料生成方法

上一次是利用srand(time(NULL))以求每一次的資料都不是同一組亂數，這次題目要求，每一次資料都必須要相同，那我們只需要將時間種子設一定數（這裡我設0）。則每一次資料皆為相同情況!

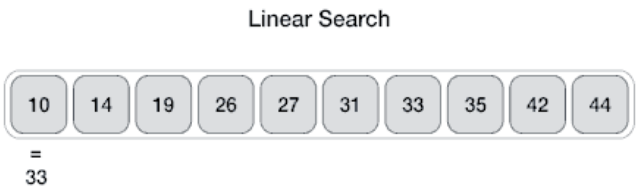
只是是陣列，我就一一比對，若遇到相同的則用while迴圈重新設一亂數，為何用while呢因為這樣可以確保重新設的亂數不是先前被使用過的數字。

ll與bst我也是一一比對，若有重複則回傳null不插入資料結構。

搜尋則先從資料結構裡隨機取一數（詳見下面解說），再依不同的搜尋方法做搜尋。

陣列

建立：利用for迴圈將資料一筆一筆放進陣列中
用linear搜尋：先從資料結構隨便取一數，一個一個比對（時間複雜度：O(n)）




```
search = arr[rand()%Bnum];
for(int j=0;j<Bnum;j++){
        if(search==arr[j]){
                break;
        }
}
```

結果：單位（ms）

1: 1205/115570

2:1045/111419

3:971/111903

4:984/112072

5:1016/111500

```
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -arr
get building num:100000
get query num:1000
arr on!
Using time of Building array 1205 ms
Using time of query array 115570 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -arr
get building num:100000
get query num:1000
arr on!
Using time of Building array 1045 ms
Using time of query array 111419 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -arr
get building num:100000
get query num:1000
arr on!
Using time of Building array 971 ms
Using time of query array 111903 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -arr
get building num:100000
get query num:1000
arr on!
Using time of Building array 984 ms
Using time of query array 112072 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -arr
get building num:100000
get query num:1000
arr on!
Using time of Building array 1016 ms
Using time of query array 111500 ms
```

二元搜尋陣列

建立：利用for迴圈將資料一筆一筆放進陣列中，再用qsort()排序
搜尋：先從資料結構隨便取一數，再用二元搜尋找配對。其概念是每次挑選中間位置的資料來比對，若該資料小於目標值，則縮小範圍為左半部，反之亦然。
(時間複雜度：O(logn))


```
void BinarySearch(int *arr,int search,int Bnum){
        int low = 0,high = Bnum-1;
        while(low<=high){
                int mid = (low+high)/2;
                if(arr[mid]==search){
                        break;
                }
                else if(arr[mid]>search){
                        high = mid-1;
                }
                else if(arr[mid]<search){
                        low = mid+1;
                }
                else return;
        }
        return;
}
```

結果：單位（ms）

1:19917/335

2:19973/288

3:19530/294

4:19718/354

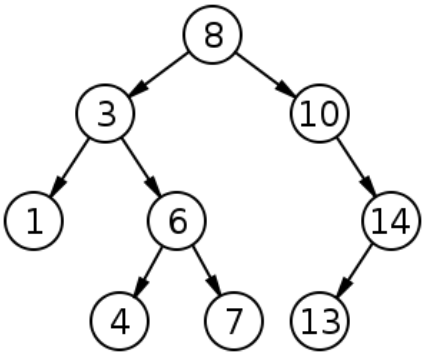
5:19754/323

```
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bs
get building num:100000
get query num:1000
bs on!
Using time of Building sorted array 19917 ms
Using time of binary search array 335 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bs
get building num:100000
get query num:1000
bs on!
Using time of Building sorted array 19973 ms
Using time of binary search array 288 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bs
get building num:100000
get query num:1000
bs on!
Using time of Building sorted array 19530 ms
Using time of binary search array 294 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bs
get building num:100000
get query num:1000
bs on!
Using time of Building sorted array 19718 ms
Using time of binary search array 354 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bs
get building num:100000
get query num:1000
bs on!
Using time of Building sorted array 19754 ms
Using time of binary search array 323 ms
```

BST

建立：先設一父節點，其規則為：
若任意節點的左子樹不空，則左子樹上所有節點的值均小於它的根節點的值；若任意節點的右子樹不空，則右子樹上所有節點的值均大於它的根節點的值；任意節點的左、右子樹也分別為二元搜尋樹；
搜尋：我們利用公式可以知道當給n個元素時最多會是log2(n)階層。我們可以利用這一點隨機設一n（rand()%資料結構長度）並從這層深的bst隨機取一數。再執行二元樹搜尋。

```
int BSTtrave(struct BSTnode* head,int Bnum){
        if(head==NULL){
                return 0;
        }
        struct BSTnode *p = head;
        for(int i=0;i<log2(rand()%Bnum);i++){
                if(p->lnode==NULL||p->rnode==NULL){
                        return p->data;
                }
                if(rand()%2==0){
                        p= p->lnode;
                }
                else {
                        p = p->rnode;
                }
        }
        return p->data;
}
```



結果：單位（ms）

1:40644/396

2:39858/422

3:40025/430

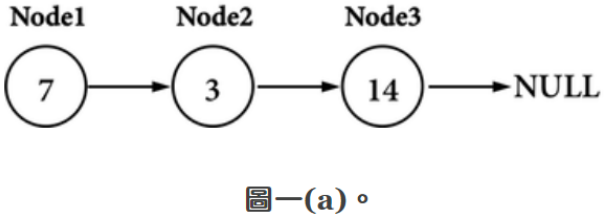
4:40067/425

5:39553/465

```
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bst
get building num:100000
get query num:1000
bst on!
Using time of Building BST 40644 ms
Using time of searching BST 396 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bst
get building num:100000
get query num:1000
bst on!
Using time of Building BST 39858 ms
Using time of searching BST 422 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bst
get building num:100000
get query num:1000
bst on!
Using time of Building BST 40025 ms
Using time of searching BST 430 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bst
get building num:100000
get query num:1000
bst on!
Using time of Building BST 40067 ms
Using time of searching BST 425 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -bst
get building num:100000
get query num:1000
bst on!
Using time of Building BST 39553 ms
Using time of searching BST 465 ms
```

LinkedList

建立：設一頭並將亂數一一設進ll裡。
搜尋：從頭開始搜尋，找到為止。



圖一(a)。

```
void LLsearch(struct LLnode* head,int search,int Bnum){
        struct LLnode* ptr = head;
        while(ptr){
                if(ptr->data == search){
                        break;
                }
                ptr = ptr->next;
        }
}
```

結果：單位（ms）

1:8639/4009

2:7249/4049

3:6872/4172

4:7414/4017

5:7707/4013

```
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -ll
get building num:100000
get query num:1000
ll on!
Using time of Building Linkedlist 8639 ms
Using time of searching Linkedklist 4009 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -ll
get building num:100000
get query num:1000
ll on!
Using time of Building Linkedlist 7249 ms
Using time of searching Linkedklist 4049 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -ll
get building num:100000
get query num:1000
ll on!
Using time of Building Linkedlist 6872 ms
Using time of searching Linkedklist 4172 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -ll
get building num:100000
get query num:1000
ll on!
Using time of Building Linkedlist 7414 ms
Using time of searching Linkedklist 4017 ms
Yongde-MacBook-Pro:HW2 yong$ ./hw2 -d 1e5 -q 1e3 -ll
get building num:100000
get query num:1000
ll on!
Using time of Building Linkedlist 7707 ms
Using time of searching Linkedklist 4013 ms
```

建立資料結構時間比較(單位：ms)

	平均時間	樣本標準差
陣列	1044	94
排序後的陣列	19778	175
鏈結串列	7536	587
BST	41882	398

分析：首先陣列是最快的因為他只要宣告一個空間放東西進去而已。排序後的陣列須較花時間。LL是因為每一次都還必須malloc一個空間才需要比陣列更多的時間。BST是因為每一次插入資料都必須比較資料與欲搜尋的資料的關係才會要更久時間（一樣要malloc）。

搜尋資料結構時間比較(單位 :ms)

	平均時間	樣本標準差
陣列線性搜尋	112492	1741
二元陣列搜尋	318	27
LL搜尋	4052	68
BST搜尋	427	24

線性搜尋最慢是因為他一個一個比對必需走過很多元素。二元搜尋是因為前面已經排序好陣列每一次比對可以減少一半不用走的冤枉路，BST其實與二元搜尋一樣。

GITHUB及參考

https://www.youtube.com/watch?v=COZK7NATh4k
https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%85%83%E6%90%9C%E5%B0%8B%E6%A8%B9
https://www.itread01.com/content/1548406293.html
https://www.geeksforgeeks.org/select-random-node-tree-equal-probability/

https://github.com/aStudentLoningToGrow/ARR-LINKEDLIST-BST-BS