**作業二 資料結構比較**

1. **測試環境**

作業系統：Microsoft Windows 10 版本20H2(OS組建19042.928)

測試環境：visual studio code

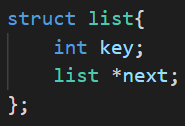
處理器：intel core i5-8265U

RAM：8GB

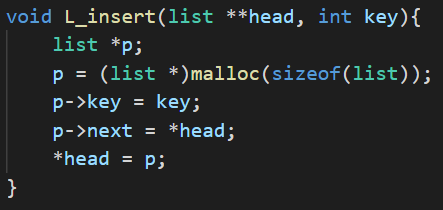
Git連結：https://github.com/kingslayer929/409410017-data-structure-compare

1. **解釋各種資料結構(linked list, array, array with binary search, BST, hash)**

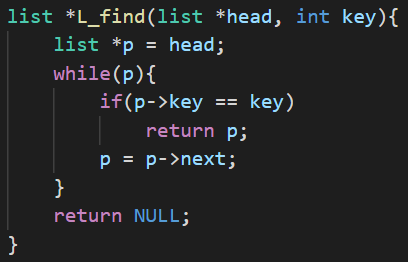
* **Linked list**



主要結構是由一個數值(key)和一個指向自身結構的指標(next)。



1. 輸入時給予串列的起頭與欲輸入的值。
2. 輸入時先取得一塊記憶體，將欲輸入的值放進此自己的key中。
3. 接著將整條串列接到自己之後，將自己改為整條串列的頭。
4. 不停重複1, 2, 3直到輸入完畢。



搜尋是否有欲搜尋的值時，給予串列開頭與欲搜尋的值。從頭開始找，不是就換下一個，直到找到後回傳位址，否則回傳空。

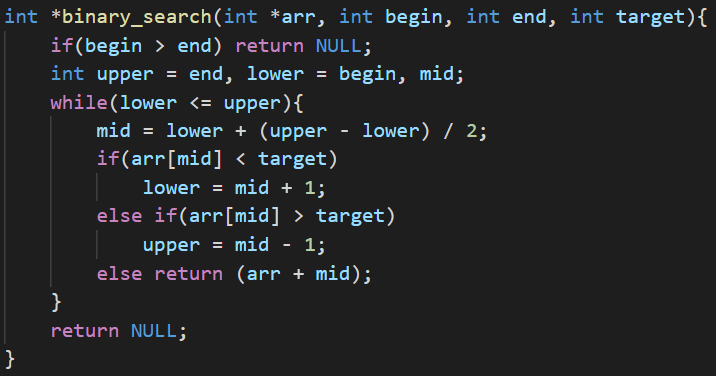
* Array

此為基本陣列操作，將值輸入，並從頭找尋是否有欲搜尋的值。

* Array with binary search

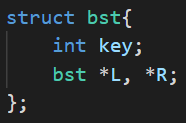
將上述的Array輸入後，進行排序。(一定要進行排序)

接著進行二分搜尋。

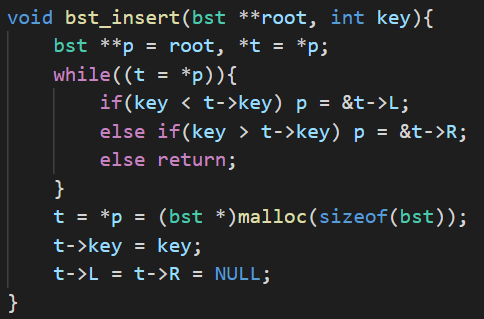


1. 給定欲搜尋陣列、陣列開頭、陣列結尾、搜尋目標。
2. 起始上界與下界為陣列開頭與結尾。
3. 中間值為上界與下界中間。從中間找，若目標比陣列中間的值小，則其值若在陣列中存在，必定在陣列中間以前，將上界改為中間值，反之則必定在陣列中間以後，將下界改為中間值。此操作可刪去目前搜尋範圍的一半，達到縮小搜尋範圍的功能。
4. 執行3直到目前中間值等於目標值，回傳目前中間值位址，否則回傳空。

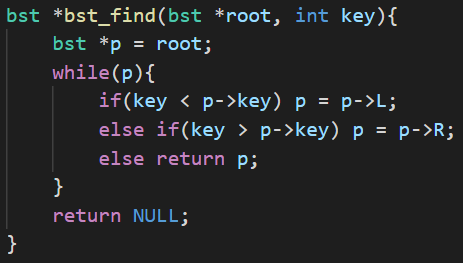
* BST (binary search tree)



主要結構是由一個數值(key)和兩個指向自身結構的指標。

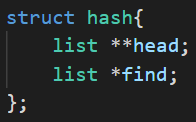


1. 輸入時給予二元搜尋樹的根與欲輸入的值
2. 與linked list不同，插入前先選擇位置，選擇方式為3, 4
3. 若此位置尚未插入，則選此位置。
4. 若此位置的值教育插入值大，位置移到左節點，反之一道右節點。
5. 重複3, 4直到找到位置，即可分配一塊空間，輸入數值。



1. 搜尋時給予二元搜尋樹的根與欲搜尋的值。
2. 與輸入時相同的方式找出此值應在的位置，若存在回傳目標位址，否則回傳空。

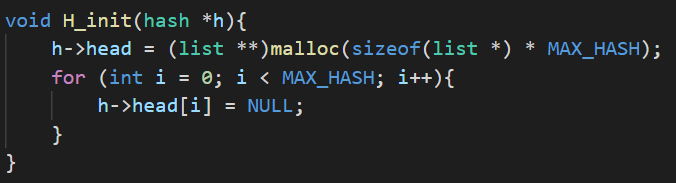
* Hash



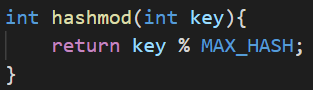
此種hash的方式為linked list陣列，並存在一個find用於儲存找尋後該值的位址。



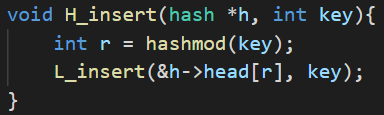
此定義hash值最多1,000,000。



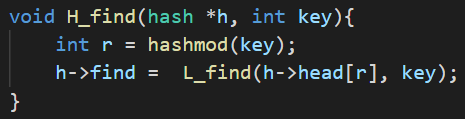
使用前必須先初始化，做出MAX\_HASH個linked list陣列，並將起始值都設為空。



此函式會以key回傳一個值，代表此key應存於哪個linked list中。



輸入時給予一個hash結構與欲輸入的值，並由hashmod得出英輸入哪條linked list。



搜尋與輸入類似，並將位址傳到結構中的find中。

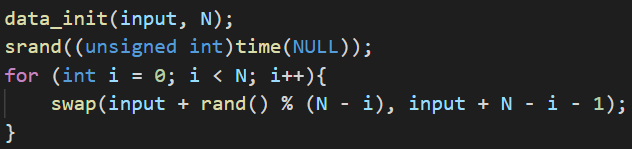
1. **亂數產生方式**

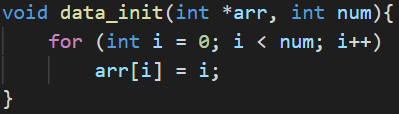


定義最大輸入與最大搜尋。



在函數之外預開兩條陣列(全域)，分別為輸入陣列與搜尋陣列。

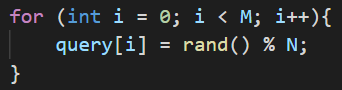




在產生輸入資料前先將輸入陣列初始化，初始化方式為第n項就等於n。

接著利用時間給予亂數種子。

生成時，分為生成區與完成區，生成區為輸入陣列的前端，初始有N個值，完成區為後端，初始有0個值。用rand()再生成區挑出其中一個值與最後一項交換，此時完成區數量+1，生成區-1，不斷重複即可得一個不重複的亂數陣列。

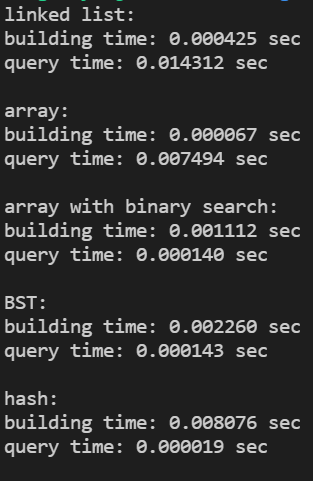


搜尋陣列的各項利用rand()產生出界於0~N-1的值供搜尋。

1. **測試**

****

輸入上述指令後，終端機上會顯示測試結果。



終端機上的測試結果。

1. **測試結果**

* **輸入**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Linked list | array | Array with binary search | BST | Hash |
| 1e4 | 0.000472 | 0.000073 | 0.001226 | 0.002154 | 0.007589 |
| 1e5 | 0.004151 | 0.000648 | 0.014151 | 0.041802 | 0.015972 |
| 1e6 | 0.039198 | 0.005567 | 0.166301 | 0.767360 | 0.125534 |

(單位：秒)

備註：表格為取三次之平均值

* **搜尋**

**資料量1e4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Linked list | array | Array with binary search | BST | Hash |
| 1e3 | 0.014774 | 0.007992 | 0.000144 | 0.000156 | 0.000021 |
| 1e4 | 0.143605 | 0.081290 | 0.001422 | 0.001630 | 0.000325 |
| 1e5 | 1.435138 | 0.780381 | 0.015486 | 0.013740 | 0.001672 |

(單位：秒)

**資料量1e5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Linked list | array | Array with binary search | BST | Hash |
| 1e3 | 0.158798 | 0.076747 | 0.000190 | 0.000378 | 0.000039 |
| 1e4 | 1.483474 | 0.740342 | 0.001830 | 0.002562 | 0.000678 |
| 1e5 | 15.08407 | 7.607624 | 0.018804 | 0.032815 | 0.002888 |

(單位：秒)

**資料量1e6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Linked list | array | Array with binary search | BST | Hash |
| 1e3 | 1.905422 | 0.776646 | 0.000455 | 0.000807 | 0.000076 |
| 1e4 | 19.05636 | 7.683893 | 0.002683 | 0.007725 | 0.000818 |
| 1e5 | 188.7827 | 75.43231 | 0.024545 | 0.078645 | 0.006918 |

(單位：秒)

備註：表格為取三次之平均值

1. **結論**

* **輸入**

資料量小時array < linked list < array with binary search < BST < hash

資料量大時array < linked list < hash < array with binary search < BST

**時間複雜度：(輸入n筆資料)**

Array O(n)

Linked list O(n)

Array with binary search O(nlogn)

BST O(nlogn)

Hash O(n)

* **搜尋**

hash < array with binary search < BST < array < linked list

**時間複雜度：(找尋一筆資料)**

Array O(n)

Linked list O(n)

Array with binary search O(logn)

BST O(logn)，最差O(n)

Hash O(1)

* **空間複雜度(n筆資料下，MAX為預開空間的量)**

Array O(MAX)

Linked list O(n)

Array with binary search O(MAX)

BST O(n)

Hash O(MAX)

總和來說，hash是在時間表現上是最優的，只是必須預開很大的空間，簡單來說就是用空間換去時間。其次是array with binary search，再來是BST，兩者最大的不同是前者輸入時不用malloc且搜尋時保證為O(logn)，不用malloc的結果依舊是以空間換取時間。最後兩名是linked list與array，用array時不如直接進行排序在搜尋(不是動態的，所以每次搜尋前都必須排序)，而linked list不適合在資料量大時做處理。