**程式設計研討專題二**

110403518 林晉宇 11/16

1. **紅黑樹**

C++ STL的map跟set容器背後就是用紅黑樹實現的，故這裡直接使用set來實作。

**時間複雜度:** Search **O(logn)** / Insert **O(logn)** / Delete **O(logn)**

**Pseudo code:**

*input n*

*for i in [0,n-1):*

*input x*

*set.insert(x)*

*input m*

*while(m--):*

*input operation, num*

*if operation==’s’:*

*if (num exist): print “Found”*

*else: print “Not Found”*

*else if operation==’d’:*

*if (num exist): print “Delete Success”, set.erase(num)*

*else: print “Delete Failed”*

*else if operation==’I’:*

*if (num exist): print “Insert Failed”*

*else: print “Insert Success”, set.insert(num)*

1. **Making Binary Search Dynamic**

**時間複雜度分析:**

1. **Search:** 遍歷每一個陣列，假設第i個陣列是滿的，其長度為2^i，因其為排序好的陣列，故可以二分搜，時間為O(i)，而i的範圍為(0,logn)，故搜尋的時間複雜度為**O(log^2(n))**。
2. **Insert:** 當有新元素加入時，直接把該元素加入進A0陣列，如果已經存在一A0陣列，則把兩個merge後形成A1，依此類推，直到不用再merge為止，假設我們需要把A0,A1….,Am-1 merge，這樣的時間複雜度為O(2^m)，而最糟情況m=k，則時間複雜度為O(n)。但我們可以發現當操作一系列的insert後，n0每次都會被改變，n1每兩次被改變…etc，每次改變代表被merge一次，所以m次的insert -> 時間複雜度可視為mO(logn) -> **O(logn)**。
3. **Delete:** 先找到不為空的最小陣列(ni!=0)，從此陣列開始找，我們可能需要k(深度，即log(n))個陣列，所以時間複雜度為**O(logn)**。

Credit: [17-2 Making binary search dynamic - CLRS Solutions (walkccc.me)](https://walkccc.me/CLRS/Chap17/Problems/17-2/)

**Pseudo code:**

*pair search(num):*

*遍歷每一層，二分搜每一層*

*If 找到num: return {num的位置}*

*else: return {-1,-1}*

*void merge(vector<int> a, vector<bool> b, h欲合併的深度):*

*if h>=現在的深度: 新增一層空陣列*

*if h層為空: 直接將a塞進h層*

*bool insert(num):*

*if該數字存在: return false*

*else if該數字被標記為刪除(false): 更改標記成true*

*else:*

*vector<int> a; a.push\_back(num);*

*vector<bool> b; b.push\_back(true);*

*merge(a,b,0) //0代表第0層*

*bool delete(num):*

*pos=search(num)*

*if 該點不存在: return false*

*else 該點存在: exist[pos]=false //把該點標記不存在*

*if pos所在層數有一半的點已經被刪除:*

*if 上一層為空: 則把當前這一半全部塞進上一層*

*else: merge(a,b,上一層)*

*main:*

*input N*

*for i~N: input x, insert(x)*

*input t*

*while(t--):*

*input operation, num*

*if operation==’s’:*

*if search(num): print “Found”*

*else: print “Not Found”*

*else if operation==’i’:*

*if insert(num): print “Insert Success”*

*else: print “Insert Failed”*

*else if operation==’d’:*

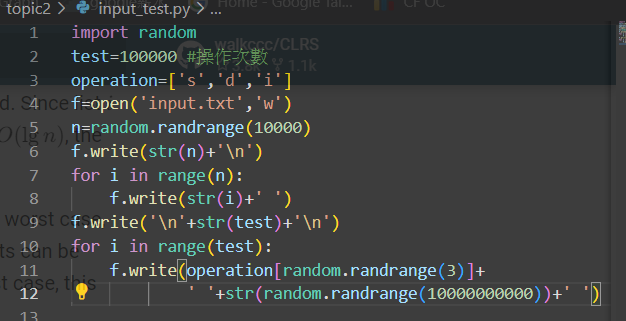
*if delete(num): print “Delete Success”*

*else: print “Delete Failed”*

*return 0*

1. **時間比較**

**產生側資**: 使用python產生側資，初始有10000個數字，再來100000個隨機操作。



1. 紅黑樹(set): 0.212秒(100000筆操作)



1. Dynamic Binary Search: 0.59秒(100000筆操作)



1. 紅黑樹(set): 2.688秒(1000000筆操作)



1. Dynamic Binary Search: 6.093秒(1000000筆操作)



1. 紅黑樹(set): 19.869秒(10000000筆操作)



1. Dynamic Binary Search: 41.766秒(10000000筆操作)



1. **結論**

從時間複雜度可以看出，dynamic binary search的search需要花比較多的時間，所以search用的越多次，時間與紅黑樹的差距越顯著，透過python產生的側資實際比較，可以發現紅黑樹與dynamic binary search的時間大概差接近三倍，所以可知c++ 的map及 set容器算十分有效率的，以後可以直接用。