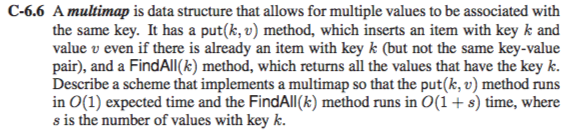
CS 600 Homework 4 | CWID 10430147 | Divyendra Patil | Username: dpatil3  
Date: 09/28/2017

**Chapter 6:**



**Solution:**  
Here, we have a multimap data structure having multiple values for a given key.

We have two methods here,

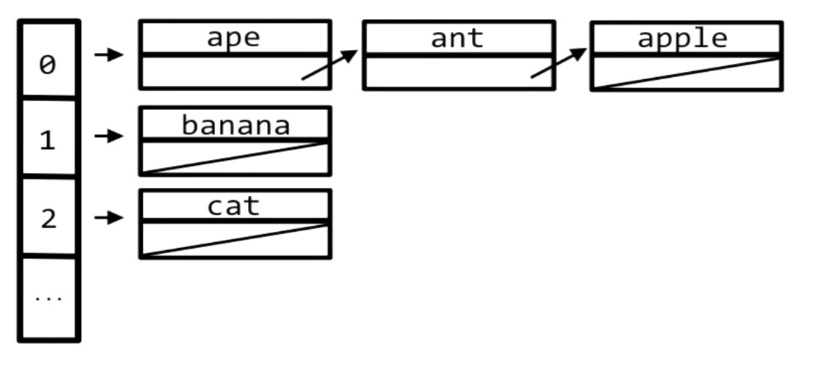
**put(k,v)** and **FindAll(k)**

put(k,v): inserts the value v at key k

FindAll(k): returns all values having key k

To implement a multimap scheme, we need to device a put(k,v) method in such a way that map operation accommodates multiple elements with the same key and not replacing elements with the same key.

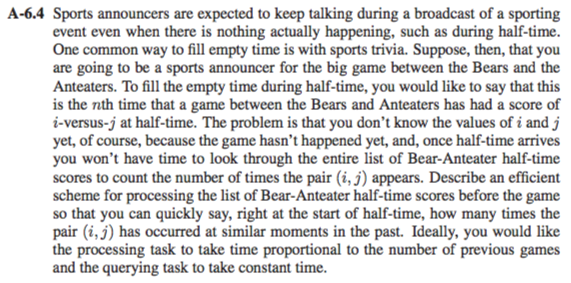
A scheme to implement a multimap such that the put(k, v) method runs in O(1) expected time and the FindAll(k) method runs in O(1 + s) time would be via utilization of a Hash Table which utilizes the link list to store multiple value at a single key(the concept of chaining is used)



As seen from the above figure, all values in the linked list are hashed with a single value in the hash table.

Hence if we want to put(k,v) it will take O(1) time as insertion in linked list where k is hashed. Even if the values map to the same key the link list can be expanded and new elements can be added in O(1) time.

Similarly, if we need to implement the FindAll(k) method it will again run in O(1+s) time, since the operation will search through the linked list where k is hashed and s is the number of elements in the linked list having k as their key.



Considering i and j, let i-versus-j represent the score of the match between Bears-versus-Anteater at half time.

To implement the above scenario, we can make use of cuckoo hash table that uses i and j as the key having a single value v which will represent the number of times the same i,j pair has occurred between Bears versus Anteaters at half time.

**Querying**:

put(k,v)

That is k = (i,j) is the key having i and j which are the scores of Bears and Anteater and v is the value of the number of matches of having the same score half time.

If the score is new then

we insert it by creating a new key-value pair having the value as 1.

If the score is already there, that is if the same key k = (i,j) is already present then

collision will occur and the value of that particular score will be incremented by 1.

For example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Match No.: | 1 | 2 | 3 | 4 | 5 |
| Key: | (1-0) | (2-1) | (2-0) | (2-3) | (1-1) |
| Value: | 1 | 2 | 2 | 1 | 1 |

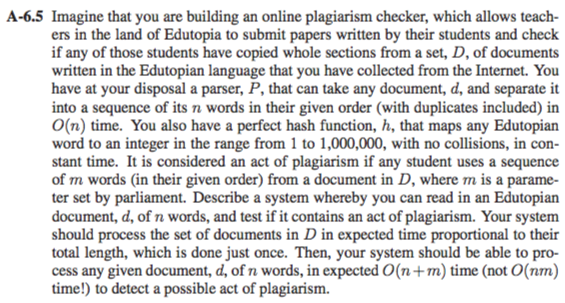
So, whenever the score is new, it is inserted with value as 1.

Suppose collision occurs at Match No. 2 and 3 which previously had the same scores at half time so the value is incremented by 1.

The announcer will simply insert the score that is the key and retrieve the value for that particular score. This value will be the number of matches that have the same score at half time. This will be retrieved in a single step.

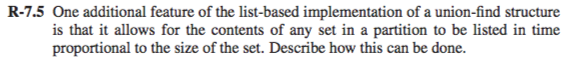
So the worst case query time complexity is O(1).

The processing time in a hash table would take a linear time as we need to traverse through the complete data set.



1. We use a hash table of size at least 2N, where N is the total length of the documents in D.
2. We have a perfect hash function h that maps any Edutopian word to an integer in the range from 1 to 1,000,000, with no collisions, in constant time.
3. Now we will map the student d of n words in O(n) time (since it’s a liner operation).
4. The words which are repeated are placed with the same keys in constant time O(1) via chaining.
5. Hence in such a case the repeated words are placed with the same key and paired together.
6. When we have m consecutive matches we can say that the student has plagiarized the document.
7. The whole process will run in O(m+n) time.

**Chapter 7:**



Considering this list-based implementation of the union-find structure below:

A={5, 1,8}, where A is the set with three nodes, 5, 1 and 8.

The set is having object A having a head node and a tail node.

The head points to the first node and the tail points to the last node.



To prove: the time required to list the nodes is proportional to the size of the set.

next returns the next node.

**Algorithm:**

Input is the object set that has the nodes 5,1,8

Output will be all the nodes.

printSet(node)

while node.next != null

print node

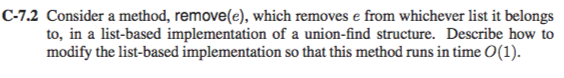
node = node.next

print node.next

To list the nodes the while loop will run according to the size of the set that is if the size is n the loop will run n times.

Hence, the time complexity is O(n) where n is the size of the set.

Hence proved that the time required to list the nodes in a list-based implementation of the union-find structure is directly proportional to the size of the set.



Consider this list-based implementation of a union-find structure.



There are three nodes, 5, 1 and 8.

A={5, 1, 8}

Now, for removing a node from the above structure, there are three positions to b considered.

In this case,

first position that is the first element 5

middle position that is 1

last element that is 8

We need to modify the list based implementation so that it runs in time O(1).

A possible way to do this would be to reorganize the linked list implementation into a doubly linked list.

Removing the first element can be done in two steps:

• Create a temporary node which will point to the same node as that of head.

• We can now move the head nodes pointer to the next node and change the heads left pointer to NULL and dispose the temporary node.

Removing the last node from a doubly linked list can be done as follows:

• Get the tail node.

• Update tail nodes previous nodes next pointer to NULL.

• Dispose the tail node.

Removing the middle node is similar to the above procedure. Find the node to be deleted, change the previous nodes next pointer to the next node of the node to be deleted.

This ensures a O(1) operation

Screen%20Shot%202017-09-28%20at%202.38.37%20PM.png

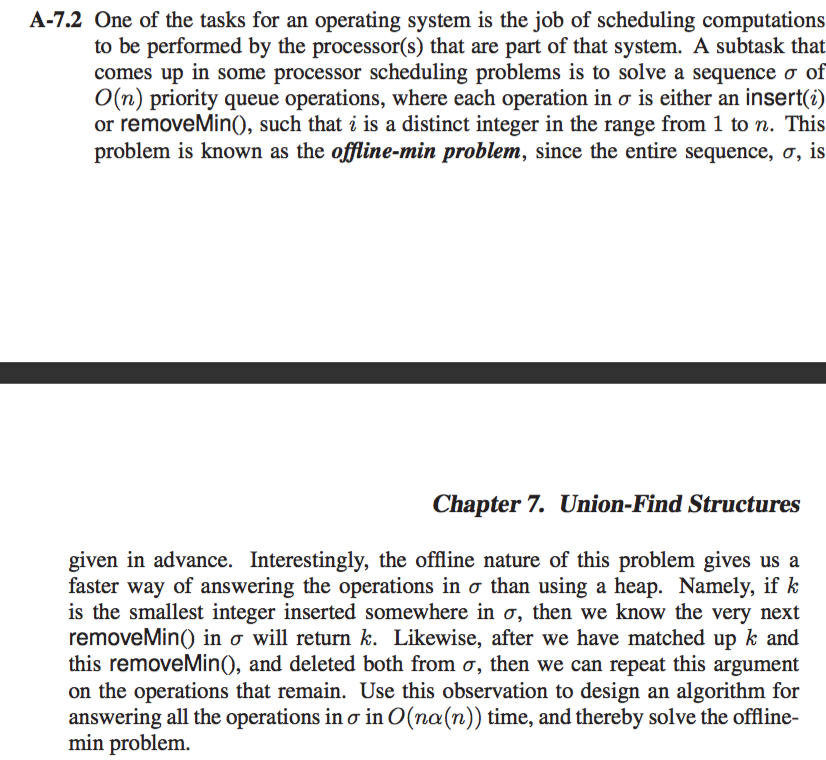
To remove the duplicates from a collection of objects i.e A,   
We need to sort A first so that it will be easier to remove the duplicates.

To sort the collection the most efficient algorithm is the merge sort which has time complexity of O(nlog n) (Make note, quick sort also has the same time complexity but also has a worst time complexity compared to merge sort , that is the reason we choose merge sort)

Now, we consider a linked list in which all the objects are placed.

After sorting, we will delete the duplicates which has O(1) time complexity since it is done using a linked list.

Thus, The total time complexity is O(nlog n) + O(1) that is O(nlog n).



Since the sequence of priority operation which needs to performed, We also know the operation which needs to be performed in advance, we can then remove or cancel certain operations as if the value to insert K is the smallest element then the RemoveMin() will remove k from the list so we cancel their operation. We assume that the operation is a sequence operation and the array is based on priority queue.

In this implementation, we take only the value which needs to be inserted and not the operation which is needed to be applied.

Consider the sequence as 5,6,4, ExtractMin, ExtractMin, 3, 7, ExtractMin, 2, ExtractMin, 8

In this example 5,6,4,3,7,2,8 are considered as the insert operation and the value corresponds to the value which needs to be inserted

Algorithm OfflineMin (sequence a)

Input: Sequence a which contains the list of operation which needs to be performed

For (i = 0; i < sizeof(a); i++)

{

If a[i] = <ExtractMin> operation then

RemoveCount++ and index = i /\*Find no of ExtractMin and increase value of removecount \*/

Else

resultarray = insert(a[i]) /\*If not ExtractMin then insert the value into another loop \*/

}

If index < sizeof(a) then /\*If ExtractMin Operation is the last in the array we check them\*/

{

Smallarr = resultarray.sub(index – RemoveCount) /\*If not last occurrence -> neglect the last element\*/

Else

{

Smallarr = resultarr /\*If Extract is the last then copy everything in another & sort \*/

Smallarr.sort(Smallarr)

}

Eradicate the number of values from resultarr according to the value of the Remove Count

resultarr = a

Finally, the existing array contains all the remaining value with or without the Extract operation

If the input is ["5","4","6","E","1","7","E","E","3","2"]  
The output should be: 4, 1, 5

Example in Javascript: (Only for reference)

function OffLineMinimum(str) {

var t, s = [], low, index = str.indexOf('E'); // find the first 'E'

while(~index){ // loop untill no 'E' left

t = str.slice(0, index); // create a subset of the array from start to current 'E'

str.splice(index, 1); // remove current 'E'

low = Math.min.apply(Math, t).toString(); // find the lowest number in the subset

str.splice(str.indexOf(low), 1); // remove the lowest number from the array

index = str.indexOf('E'); // find the next 'E'

s.push(low); // push lowest number to return array

}

return s;

};

Reference Link: http://www.codingforums.com/javascript-programming/354115-offline-minimum-problem-help-sorry.html