***Chapter 6***

***C-6.6*** A *multimap* is data structure that allows for multiple values to be associated with

the same key. It has a put (*k, v*) method, which inserts an item with key *k* and value *v* even if there is already an item with key *k* (but not the same key-value pair), and a FindAll(*k*) method, which returns all the values that have the key *k*. Describe a scheme that implements a multimap so that the put (*k, v*) method runs in *O* (1) expected time and the FindAll(*k*) method runs in *O* (1 + *s*) time, where

*s* is the number of values with key *k*.

**Answer:** There are two methods put(k,v) and FindAll(k). put (k,v) is for inserting k value at key k and FindAll returns all values that has key k.

There is a multimap data structure as multiple values are assigned to a key. To implement the scheme the put method is used that the map can accommodate multiple elements with the same key and not replace the elements that has the same key.

The method put runs in O(1) time and FindAll runs in O(1+s) over here has table can be itlized to link the list which can store the multiple value at a single key.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Search Value | → | Hash Function | Key | → | Value | → | Value | → | Value |
| 0 | → | Abcd | → | Adefg | → | Ahgi |
| 1 | → | Befb | → | Bjb | → | Bjkbui |
| 2 | → | Cubuj | → | Cuguj | → | Cuyvhj |
| 3 | → | Djubukj | → | Dihk | → | Djk |
| .. | → | … | → | … | → | … |
| 6 | → | Glknkl | → | Guihgkjh | → | gujkk |

In the above image/table all the values of the linked list are hashed into single value in the has table.

So, when the put (k,v) method is called it takes O(1) amount of time as insertion where the k is hashed.

For the method FindAll(k) method would run in O(1+s) the operation would search through the entire linked list where k I hashed and s would be the number of elements in the list whose key is k.

***A-6.4*** Sports announcers are expected to keep talking during a broadcast of a sporting

event even when there is nothing actually happening, such as during half-time.

One common way to fill empty time is with sports trivia. Suppose, then, that you

are going to be a sports announcer for the big game between the Bears and the

Anteaters. To fill the empty time during half-time, you would like to say that this

is the *n*th time that a game between the Bears and Anteaters has had a score of

*i*-versus-*j* at half-time. The problem is that you don’t know the values of *i* and *j*

yet, of course, because the game hasn’t happened yet, and, once half-time arrives

you won’t have time to look through the entire list of Bear-Anteater half-time

scores to count the number of times the pair (*i, j*) appears. Describe an efficient

scheme for processing the list of Bear-Anteater half-time scores before the game

so that you can quickly say, right at the start of half-time, how many times the

pair (*i, j*) has occurred at similar moments in the past. Ideally, you would like

the processing task to take time proportional to the number of previous games

and the querying task to take constant time.

**Answer:** A hash table can be used that has I and J which are the scores of Bears and Anteater. The same put method that is used in the precious question can be used. put(s,v) where s = (i,j) the key s and v is the value of the number of matches of having the same score half time.

Now, if there is a new score then it can be inserted by creating a new key-value pair that would have the value 1 and if the score remains constant then the value the score will be incremented by 1.

Match No. 1

Key s (1-0)

Value v 1

Match No. 2

Key s (2-1)

Value v 2

Match No. 3

Key s (2-0)

Value v 2

Match No. 4

Key s (2-3)

Value v 1

Here whenever the new score is added it is added as the value 1, and if there is a collision for example what happens at number 2 and number 3 the previous score first has 1 value added to it.

So, the score here can be just inserted, the value will but the number of matches that have the same score at half time. Here the Worst-case scenario is expected to be O (1). The pair (in, jn) would be the key and value can be count n.

***A-6.5*** Imagine that you are building an online plagiarism checker, which allows teachers

in the land of Edutopia to submit papers written by their students and check

if any of those students have copied whole sections from a set, *D*, of documents

written in the Edutopian language that you have collected from the Internet. You

have at your disposal a parser, *P*, that can take any document, *d*, and separate it

into a sequence of its *n* words in their given order (with duplicates included) in

*O*(*n*) time. You also 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. It is considered an act of plagiarism if any student uses a sequence

of *m* words (in their given order) from a document in *D*, where *m* is a parameter

set by parliament. Describe a system whereby you can read in an Edutopian

document, *d*, of *n* words, and test if it contains an act of plagiarism. Your system

should process the set of documents in *D* in expected time proportional to their

total length, which is done just once. Then, your system should be able to process

any given document, *d*, of *n* words, in expected *O*(*n*+*m*) time (not *O*(*nm*)

time!) to detect a possible act of plagiarism.

**Answer:**

* A hash function(h) is required to that would map each key k in the map to the integer in the range [0, N-1], where N is the capacity of the array. The lookup table is used to approach to arbitrary keys. Or a hash table with size 2N where n is the lengths of documents in D.
* Hash value is used as an index into the array A instead of the key k
* The items is supposed to be inserted so (k,v) is inserted in A[h(k)]
* It is possible that h(k) = h(l) where k =l two distinct keys k and l can be in the map at the same location with same has value j where there would be a collision. So, Hash function h has to be random which would minimize the expected number of collusions for hash value j.
* Each key, k is a tuple of integers, (x1, x2, …, xd) with each xi being an integer in the range [0, M-1] for some M and d=1,000,000 value. If there are no collisions in the constant time then the words that are repeated are placed with the same keys in constant time O (1) via the chaining. These repeated words then are paired together and the consecutive matches would say if the student has plagiarized the document.
* Each key k is a nonnegative integer which could be possibly very large.
* The runtime of the whole process would be O (m+n) time.

***Chapter 7***

***R-7.5*** One additional feature of the list-based implementation of a union-find structure

is that it allows for the contents of any set in a partition to be listed in time

proportional to the size of the set. Describe how this can be done.

Answer: Suppose this is a list base implementation of the union find structure

A = {7,2,9} and B= {4,1,8}

Here the set has object a with a head node and tail node. The head points to the first node and the tail points to the last node. Each set can be traversed starting from the header pointer.

A

7 2 9

B

4 1 8

Algorithm

Input is the object set, and the output would be all the nodes.

printSet(node)

while node.next != null

print node

node = node.next

print node.next

The loop will run as per the size of the set to list all the nodes of the set so, if the size is n then the loop will run n times. So, here time complexity is O(n)

***C-7.2*** Consider a method, remove(*e*), which removes *e* from whichever list it belongs

to, in a list-based implementation of a union-find structure. Describe how to

modify the list-based implementation so that this method runs in time *O* (1).

**Answer:**

Let’s use the same example as previous question as set

A

7 2 9

It would take O(n) amount of time to find the position of the target node the node that is supposed to be removed so, in this example run time is O (1).

First a temporary node is created which would point to the same node as that of head and then the head nodes directs to the next node and change the heads left pointer to null and dispose the temporary created node. - for the first node

For the last node the tail node is pulled, it’s previous node’s next pointer to Null and the tail node is disposed. For the middle node similar procedure, node target must be found, the precious node’s next pointer must be updated to next node of the node to be removed.

***C-7.4*** Let *A* be a collection of objects. Describe an efficient method for converting *A*

into a set. That is, remove all duplicates from *A*. What is the running time of this method?

**Answer:** To sort the collection A the efficient method is to use merge sort, that has the time complexity of O (nlog n) it is better option compare to quick sort considering the worst-case scenario. If the range is known bucket sort would only take O(n) and then it can be traversed while removing the duplicates.

After sorting duplicated are deleted which would take O (1) time complexity as it is done using the linked list. O (n log n) + O (1) is O (n log n).