COMP 6481: Programming and Problem Solving

Tutorial 9:

Hashing, Double Hashing, Sorting

MAP ADT

- A Map is an ADT that allows us to store values based on unique keys.
- These entries are typically called key-value pairs.
- The keys are required to be unique so that each key only points to one value.
- This is similar to how an array works except that instead of integer indexes, we use a more generic key as the "index"

MAP ADT - ORDERED MAP

- An ordered map stores key-value pairs in an ordered search table.
- An ordered search table is an implementation of a Map using an ordered ArrayList.
- Binary Search is a classic algorithm to locate an entry in the table. It runs in O(log n) time.

Hash Tables

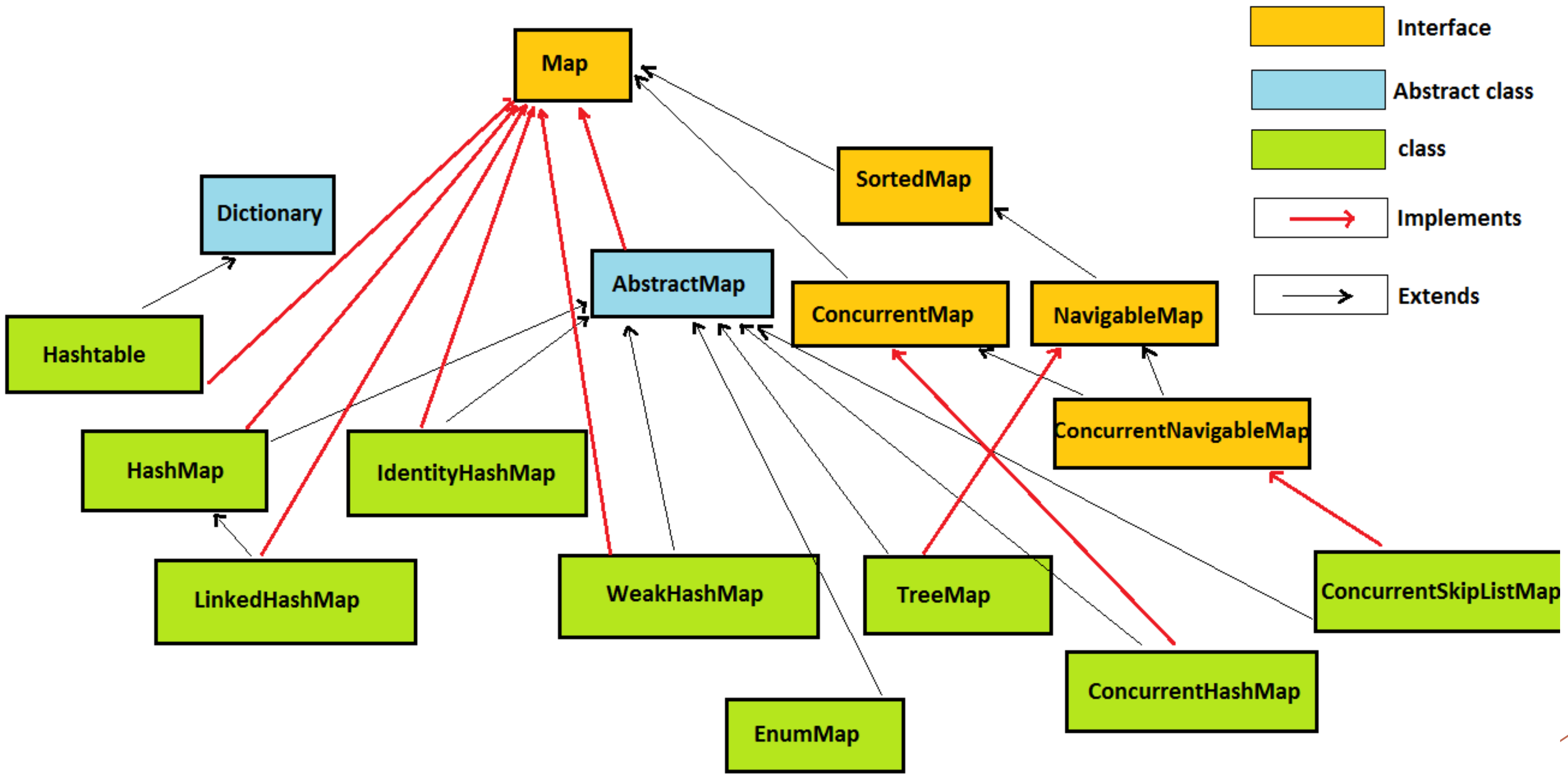
- A hash table is an efficient means to store a map.
- A Hash table consists of two components:

1) Bucket Array:
An array of a generally fixed size where each entry can
be thought of as a "bucket“ (list) that contains a set of key/value pairs.

2) Hash Function:
A function that maps the generic key type to an integer to
use as an index for the Bucket Array. The result of the hash
function is called a hash value.

Java Data structure

- HashMap is implemented as a hash table, and there is no ordering on keys or values.
- TreeMap is implemented based on red-black tree structure, and it is ordered by the key.
- LinkedHashMap preserves the insertion order
- Hashtable is synchronized, in contrast to HashMap. It has an overhead for synchronization. This is the reason that HashMap should be used if the program is thread-safe.


Map
SortedMap
Dictionary
AbstractMap
ConcurrentMap
NavigableMap
Hashtable
HashMap
IdentityHashMap
ConcurrentNavigableMap
LinkedHashMap
WeakHashMap
TreeMap
ConcurrentSkipListMap
ConcurrentHashMap
EnumMap
Interface
Abstract class
class
Implements
Extends

HASH FUNCTIONS -- COLLISIONS

- One issue with hash tables is how well the Hash Function behaves. That is to say, how well the

keys map to integers.

- When two keys share the same hash value (result of the hash function), we get a collision.
- A good hash function minimizes collisions under most conditions.
- The way a hash table implementation handles collisions has an impact on the running time complexity of functions relying on the hash table.

Collision Handling

- **Separate Chaining:** each Bucket A[i] stores a small map (list)
- **Linear Probing:** A[i+1 mode N] i=h(k) if A[i+1]occupied try A[i+2] mode N
- **Quadratic Probing:** A[i+f(j) mode N] f(j)=j^2 j=0,1,2,3,..
- **Double Hashing:** A[i+f(j) mode N] f(j)=j.h’(k) j=1,2,3

Problem Solving

- Assume an 11 entry hash table
- Use the hash function h(i) = 2i + 5 mod 11
- Insert the keys: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5
- Draw the contents of the hash table given that for collisions:

. Chaining is used

. Linear Probing is used

. Quadratic Probing is used

. Double Hashing is used with h'(k) = 7 - (k mod 7)

Problem Solving

- Consider a hash table of size 7 with hash function (h(k) = k mod 7). Draw the table that results after inserting, in the given order, the following values: 19,26,13,48,17 for each of the three scenarios below :

  1) when collisions are handled by separate chaining

  2) when collisions are handled by linear probing

  3) when collisions are handled by double hashing

using a second hash function:

h’(k) = 5 – (k mod 5)

Problem Solving

- Let H be a hash table where collisions are handled by Linear Probing and where re-hashing is used each time the load factor (number of item in the table divided by the size of the table) exceeds ½. We assume that the initial size of H is 2 and that re-hashing doubles the size of the table. After inserting 10 items with different keys, what is the size of the hash table H?

Problem Solving

- Assume an M entry hash table which needs to store N keys.
- Use the hash function h(i) = i mod M
- What is the worst-case search time?
- Would you use this for time critical applications?

Problem Solving

- Consider an initially empty hash table of size M and hash function h(x) = x mod M. In the worst case, what is the time complexity to insert n keys into the table if separate chaining is used to resolve collisions (without re-hashing) ? Suppose each entry (bucket) of the table stores and ordered linked list. When adding a new element to unordered linked list, such as element is inserted at the beginning of the list.
- What is the answer if the linked list are ordered ?
- What is the answer if the collisions are resolved using linear probing , and n <= M/2

Problem Solving

- Assume a 2D array A with a size of n x n.
- This array only contains 1s and 0s.
- All the 1s are before all the 0s in each row
- Describe an algorithm to count all the 1s in A that runs in O(n log n) and not O(n^2)
- example A with n = 4 might look like:

```
1 1 1 0
0 0 0 0
1 0 0 0
1 1 1 1
```

Quicksort: outline

Recursive method:

Input: array, firstindex, lastindex

1. Check the stopping case: firstindex<lastindex
    1. Find the splitpoint : partition□ Most important point NEXT SLIDE!!!!
    2. Recursion on left part
    3. Recursion on Right part

The Quicksort : algorithm

Partition: return the pivot position

1) Choose a pivot

2) Set a left pointer and right pointer

3) Compare the left pointer element (lelement) with the pivot and the right pointer element (relement) with the pivot.

4) Check if lElement<pivot and rElement>pivot:

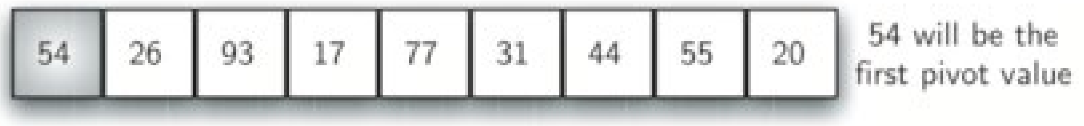
   a. If yes, increment the left pointer and decrement the right pointer

   b. If not, swap the lElement and rElement

5) When left >= right, swap the pivot with either left or right pointer.

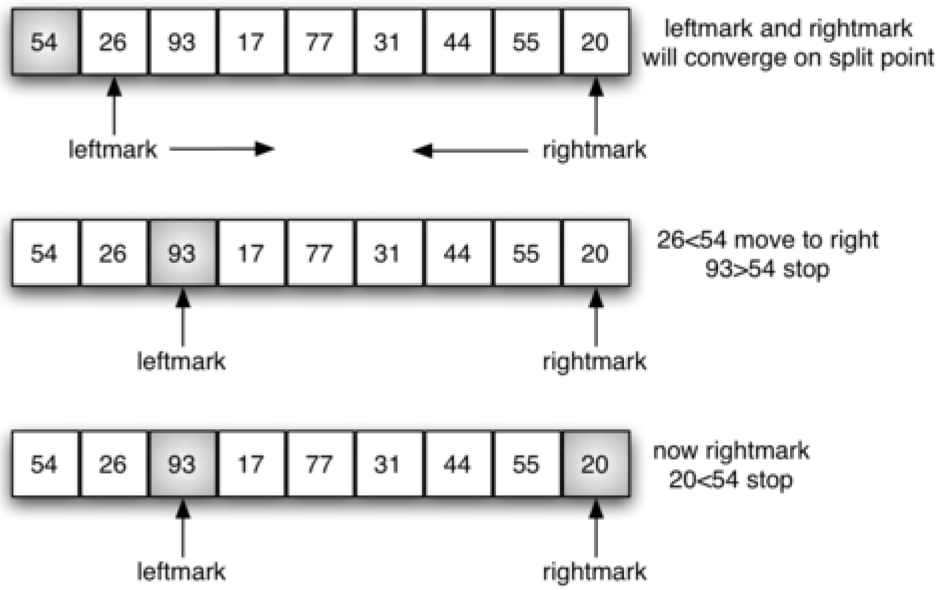
Partition algorithm: example

1. Choosing the pivot:

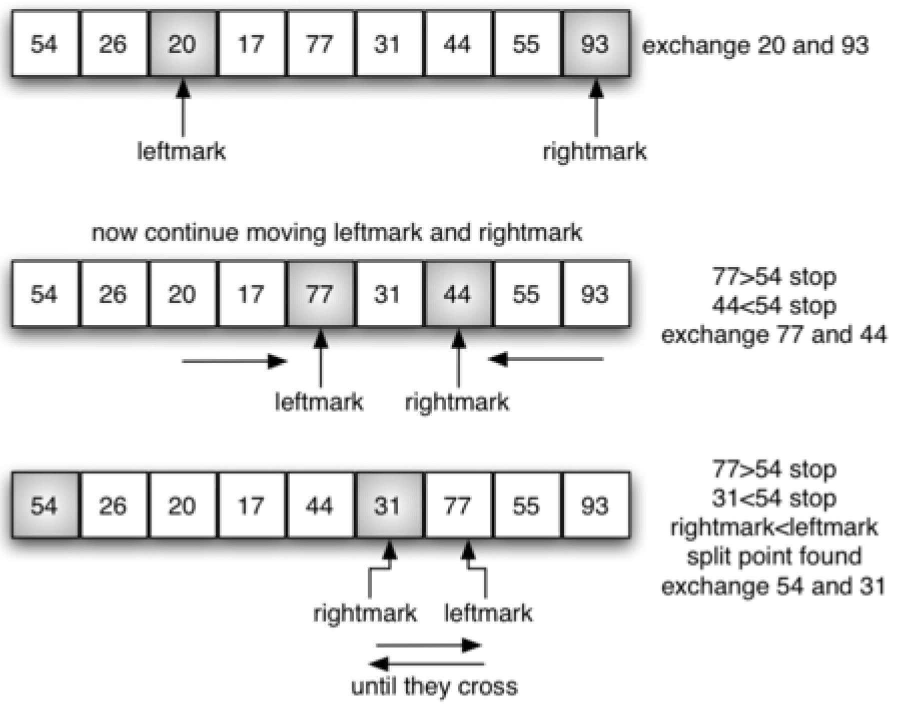


1. Moving through the array to find the last position of the pivot: the partition

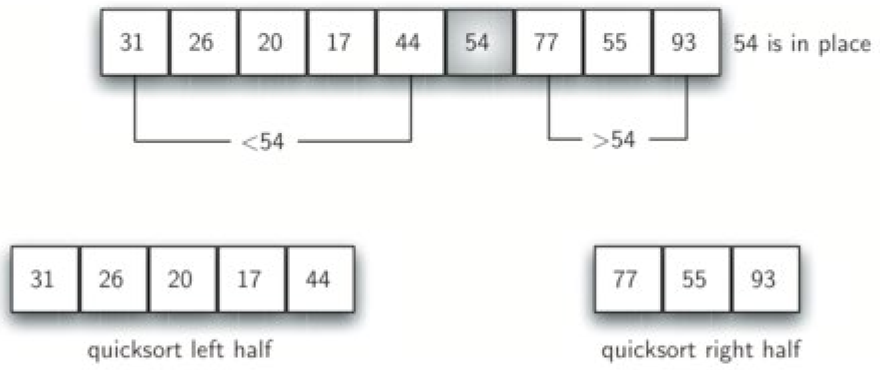
Partition: cont'd



Partition: cont'd

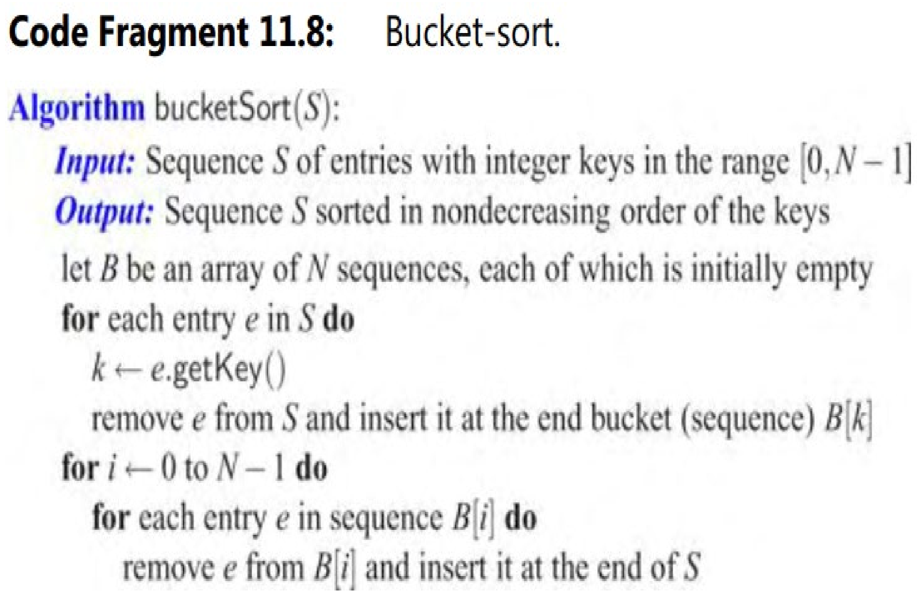


Partition: end



Bucket-sort

- Consider a sequence S of n entries whose **keys are integers in the range [0, N−1]**, for some integer N ≥ 2, and suppose that S should be sorted according to the keys of the entries. The crucial point is that, because of the **restrictive assumption** about the format of the elements, we can avoid using comparisons



Code Fragment 11.8: Bucket-sort.

```
Algorithm bucketSort($S$):
    Input: Sequence $S$ of entries with integer keys in the range $[0, N-1]$
    Output: Sequence $S$ sorted in nondecreasing order of the keys
    let $B$ be an array of $N$ sequences, each of which is initially empty
    for each entry $e$ in $S$ do
        $k \leftarrow e$.getKey()
        remove $e$ from $S$ and insert it at the end bucket (sequence) $B[k]$
    for $i \leftarrow 0$ to $N-1$ do
        for each entry $e$ in sequence $B[i]$ do
            remove $e$ from $B[i]$ and insert it at the end of $S$
```

Example:
For simplicity, consider the key in the range 0 to 9.
Input data: $(1, v_1)$, $(4, v_2)$, $(1, v_3)$, $(2, v_4)$, $(7, v_5)$, $(5, v_6)$, $(2, v_7)$

Sorting with inbuilt java functions

- Arrays.sort(arr);
- Arrays.sort(int[] a, int fromIndex, int toIndex)
- Arrays.parallelSort(toSort); // with java 8
- Collections.sort(list);

Java’s implementation has started using Timsort.

How to Sort Map by keys?

How to sort by value?

Problem Solving

- Given an array of size n, find all elements in array that appear more than n/k times. For example, if the input arrays is {3, 1, 2, 2, 1, 2, 3, 3} and k is 4, then the output should be [2, 3]. Note that size of array is 8 (or n = 8), so we need to find all elements that appear more than 2 (or 8/4) times. There are two elements that appear more than two times, 2 and 3.

Problem Solving

- You are given a set of n real numbers and another real number x. Describe an O(*nlogn*) time algorithm that determines whether or not there exists 2 elements in S whose sum is exactly x.

Problem Solving

You are given an array of n+2 elements. All elements of the array are in range 1 to n. And all elements occur once except two numbers which occur twice. Find the two repeating numbers.

For example, array = {4, 2, 4, 5, 2, 3, 1} and n = 5

The above array has n + 2 = 7 elements with all elements occurring once except 2 and 4 which occur twice. So the output should be 4 2.

Problem Solving

Suppose we are given an n-element sequence S such that each element in S represents a different vote for president, where each vote is given as an integer representing a particular candidate. Design an $O(n\log n)$ time algorithm to see who wins the election S represents, assuming the candidate with the most votes wins (even if there are $O(n)$ candidates).