CS 61B Spring 2017 Guerrilla Section 5 Solution

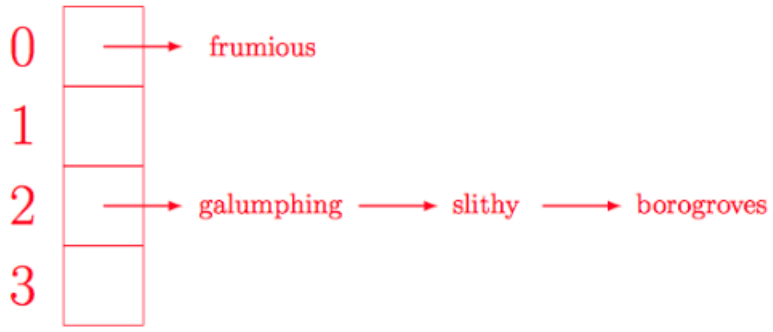
18 March 2017

Directions: In groups of 4-5, work on the following exercises. Do not proceed to the next exercise until everyone in your group has the answer and *understands why the answer is what it is*. Of course, a topic appearing on this worksheet does not imply that the topic will appear on the midterm, nor does a topic not appearing on this worksheet imply that the topic will not appear on the midterm.

1 Hashing Mechanics

Suppose we insert the following words into an initially empty hash table, in this order: **galumphing, frumious, slithy, borogroves, mome, bandersnatch**. Assume that the hash code of a String is just its length (note that this is not actually the hash code for Strings in Java). Use external chaining to resolve collisions. Assume 4 is the initial size of the hash table's internal array, and double this array's size when the load factor is equal to 1. Illustrate this hash table below with a box-and-pointer diagram.

After the first 4 insertions, the hash table looks like this (note that the order of the terms in each linked list is not important and depends on the implementation):



Before the next insertion, the array is resized. After all insertions are completed, the hash table looks like this (shown on next page):

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

2 Hash Code Design

Describe a potential problem with each of the following:

1. An implementation of the hashCode method of the String class that simply returns the length of the string (i.e. the hash code used in problem 1).
   This will likely cause many collisions since many distinct strings have the same length. Since the efficiency of hash tables depends on a low number of collisions, this would result in inserting, removing, and finding objects in the hashtable to be slow.

2. An implementation of the hashCode method of the String class that simply returns a random number each time.
   Hash functions must be deterministic, so this hashCode is not even valid. If random numbers are used, then hashCode may return different values for the same object when it is called repeatedly.

3. Overriding the equals method of a class without overriding the hashCode method.
   If equals is overridden and hashCode is not, then it is possible that two objects will be equal (according to the equals method) but have different hashCodes. For example, the Object class hashCode returns different values for all objects, regardless of whether or not theyre equal. This may cause a HashSet to report that an object is not prevent even if some identical object is in the HashSet.

4. Overriding the hashCode method of a class without overriding the equals method.
   This causes the same problem as part (c).

5. Modifying an object after inserting it into a HashSet.
   This may prevent us from being able to find the object again since the HashSet uses the objects hashCode to check if the object is present (and if the object is modified, its hashCode may also change).

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

3 Hash Table Runtimes

1. What is the worst-case runtime for inserting a single entry into a HashMap containing N elements? You may assume that hashing a key requires a constant amount of time. Provide your answer using Θ notation.
   $\Theta(N)$

2. Describe a situation in which we would achieve the above runtime.
   An example of a situation that would achieve this runtime is if all elements hashed into the same bucket, so that we essentially just have a linked list inside one bucket. This can occur when our hashing function is poorly designed.

3. After finishing Lab 9, Janice decides to create her own hash table implementation: `SmallMap`. In order to avoid costly resizing operations, `SmallMap`'s internal array does not resize; it has a fixed length of 1,024. What is the best-case runtime for insertion into a `SmallMap` containing N elements? The worst-case runtime? Assume that the N elements are distributed uniformly. Provide your answer using Θ notation.
   Best case: $\Theta(N)$
   Worst case: $\Theta(N)$

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

4 Writing Hashcode Functions

Implement a `hashCode` function for the class `Politician`. Try to design your hash code to minimize collisions (e.g. George H. W. Bush and George W. Bush should hash to different buckets). Your hash code does not need to be a perfect hash.

```
public class Politician {
    String firstName;
    String lastName;
    int birthYear;

    @Override
    public boolean equals(Object o) {
        if(this.getClass() != o.getClass()) {
            return false;
        }
        Politician other = (Politician) o;
        return other.firstName.equals(this.firstName) &&
                other.lastName.equals(this.lastName) &&
                 other.birthYear == this.birthYear;
    }

    @Override
    public int hashCode() {
        return this.birthYear + this.firstName.hashCode() + this.lastName.hashCode();
    }
}
```

Now, write a hashcode for a "binary" string which only contains characters that are 0 and 1.

```
public class BinString {
    String binStr;

    @Override
    public boolean equals(Object o) {
        return this.getClass() == o.getClass() && ((BinString) o).binStr.equals(this.binStr);
    }

    @Override
    public int hashCode() {
        int total = 0;
        String s = this.binStr;
        for (int i = 0; i < s.length; i++) {
            char c = s.charAt(i);
            total = 2 * total + (c - '0');
        }
        return total;
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

5 Assorted Heap Questions

(a) Describe a way to modify the usual max heap implementation so that finding the minimum element takes constant time without incurring more than a constant amount of additional time and space for the other operations.
Simply add a variable that keeps track of the minimum value in the heap. When inserting a new value, simply update this variable if the new value is smaller than it. Since the max heap only supports removing the largest element, rather than arbitrary elements, the minimum element will only be removed when the heap becomes empty, at which point we will need to reset the variable keeping track of the minimum value.

(b) In class, we looked at one way of implementing a priority queue: the binary heap. Recall that a binary heap is a nearly complete binary tree such that any node is smaller than all of its children. There is a natural generalization of this idea called a d-ary heap. This is also a nearly complete tree where every node is smaller than all of its children. But instead of every node having two children, every node has d children for some fixed constant d.

i. Describe how to insert a new element into a d-ary heap (this should be very similar to the binary heap case). What is the running time in terms of d and n (the number of elements)?
To insert, we add the new element on the last level of the tree and then bubble it up, much as in a binary heap. When bubbling up, we only need to compare the node to its parent. Since the tree has $\Theta(\log_d(n))$ levels and we have to do at most one comparison (compare the node to its parent) and one swap at each level, insertion takes $\Theta(\log_d(n))$.

ii. What is the running time of finding the minimum element in a d-ary heap with n nodes in terms of d and n?
The minimum element is simply the root, just as with a binary min-heap. So finding it takes $\Theta(1)$ time.

iii. Describe how to remove the minimum element from a d-ary heap (this should be very similar to the binary heap case). What is the running time in terms of d and n?
To remove the minimum element, we first replace it with the last element on the last level of the heap and then bubble this element down, just as in a binary heap. When bubbling a node down, we have to compare the node to all of its children to determine if it is larger than any of them and to find the smallest one (since we must swap with the smallest child to preserve the heap property). So at each level we have to do at most $\Theta(d)$ comparisons and 1 swap. Since there are $\Theta(\log_d(n))$ levels, removing the minimum takes $\Theta(d \log_d(n))$ time.

(c) Suppose a value in a max heap were changed. To recreate the heap to reflect the modified value, would you bubble the value up, bubble it down, both, or neither? Briefly defend your answer.
If the value is now greater than its parent, bubble it up. If the value is now less than one of its children, bubble it down.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

6 Zork Sort

Suppose you are given an array A with n elements such that no element is more than k slots away from its position in the sorted array. Assume that $k > 0$ and that k, while not constant, is much less than n ($k \ll n$).

(a) Fill in the blanks such that the array A is sorted after execution of this method. The important operations on a `PriorityQueue` are `add(x)`, `remove()` (remove smallest), and `isEmpty()`. Your solution should be as fast as possible.

```
public static void zorkSort(int[] A, int k) {
    int n = A.length;
    int i = 0;
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    while (i < k) {
        pq.add(A[i]);
        i += 1;
    }
    while (i < n) {
        A[i - k] = pq.remove();
        pq.add(A[i]);
        i += 1;
    }
    while (!pq.isEmpty()) {
        A[i - k] = pq.remove();
        i += 1;
    }
}
```

(b) What is the running time of this algorithm, as a function of n and k? Justify your answer.
$\Theta(n \log(k))$

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

7 Linguistics

President Trump's speaking style has attracted a great deal of attention in recent years. As curious students, we're interested in learning what Trump's favorite words are. Consider this problem: we'd like to determine the top k most common words found in the list of all n words that The Donald has ever spoken. Describe how you would solve this problem efficiently and which data structure(s) you would use. Hint: we want our algorithm to return Trump's k favorite words in $O(n \log(k))$. You may assume $k \ll n$.

We'll find Trump's favorite words using a min-heap of k words, along with a hash table of words to frequencies.

First, we will need to iterate over Trump's words and build our hash table. When a new word is encountered, it is added to the hash table with an initial frequency of 1. When an existing word is encountered, increment its frequency.

From here, we can retrieve the top k trending words using a min-heap. Iterate over the words in the hash table:

1. If the min-heap still has fewer than k elements, add the word to the heap.
2. Else, peek at the min-heap. If the least frequent word in the heap is less frequently used than the current word, remove from the heap and add the current word.

Building the hash table takes $\Theta(n)$ time and retrieving the k most frequent words takes $O(n \log k)$. Therefore, our total runtime is $O(n \log k)$.

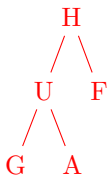
STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

8 Hidden Message

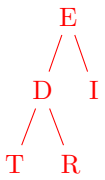
(a) Given the post-order sequence (G A U F H) and in-order sequence (G U A H F), construct the tree and find the pre-order sequence.

Tree:

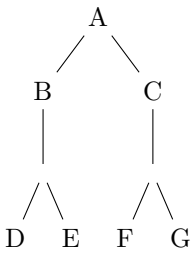


Pre-Order Sequence: (H U G A F)

(b) Given the post-order sequence (T R D I E) and pre-order sequence (E D T R I), construct the tree and find the in-order sequence. For this question you can assume the tree is full (a full tree means that each node will always have two children).



(c) Suppose we are searching for nodes in the following tree:



To search this tree, we may use either of the following two traversal methods: preorder depth-first and level-order. Assume that both traversals will break ties by going left. Which method will require us to look at the fewest nodes, assuming we're looking for node D? What about for nodes C and G? Keep in mind the answer may be a tie.
A depth-first traversal would find D most quickly. A level-order traversal would find C most quickly. Each traversals would find G equally quickly.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!