Happy M/T Studying! Stay dry !!

CSE 373 Autumn 2019 — GS 4

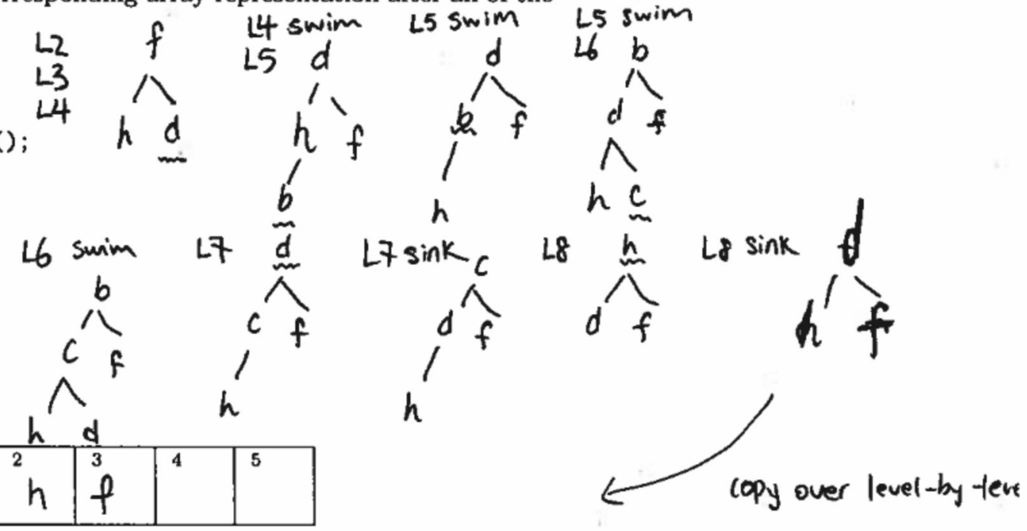
October 17, 2019

With thanks to CS 61B staff for authoring these questions.

1 A Heap of Operations

Assume that we have a binary min-heap (smallest value on top) data structure called `Heap` that stores characters, and has properly implemented `insert` and `removeMin` methods. Draw the heap and its corresponding array representation after all of the operations below have occurred:

```
1 Heap<Character> h = new Heap<>();
2 h.insert('f');
3 h.insert('h');
4 h.insert('d');
5 h.insert('b');
6 h.insert('c');
7 h.removeMin();
8 h.removeMin();
```



We leave index 0 empty →

0	1	2	3	4	5
→	d	h	f		

Tip: When doing the question by hand, you can draw out the tree node representation.

2 In The Heap of The Moment

You are tasked to implement a max-heap data structure of integers using only a min-heap of integers. Could you complete the task? If so, fill in the circle next to "Yes" and describe your approach and how you would implement the max-heap's methods (`insert`, `removeMax`, and `findMax`). If not, fill in the circle next to "No" explain why it's impossible.

[✓ Yes] No

Sample solution: when writing numbers, store negated version. When reading using removeMin() and findMin(), negate returned value prior to returning.

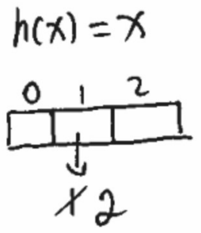
3 Going, Going, Gone

For each of the following scenarios, mark **Always**, **Sometimes**, or **Never** and explain your reasoning.

(a) When you modify a key that has been inserted into a HashMap, will you be able to retrieve that entry again?

○ Always [✓ Sometimes] ○ Never

If the new element is expected to go into a different bucket then you won't find it there the next time!



(b) When you modify a value that has been inserted into a HashMap, will you be able to retrieve that entry again?

[✓ Always] ○ Sometimes ○ Never

We don't consider the value at all when hashing!

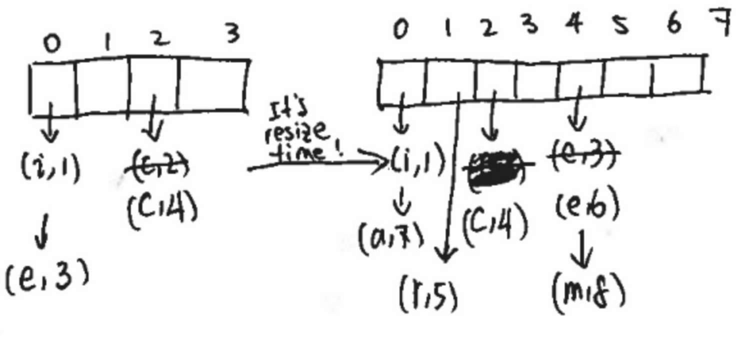
4 Filling The Buckets

Assume we have a HashMap<Character, Integer> that starts with 4 buckets, resizes by doubling the number of buckets if the insert would cause the load factor to exceed 0.75, and implements external chaining using linked lists. Draw the HashMap as we've seen in lab after inserting: ('i', 1), ('c', 2), ('e', 3), ('c', 4), ('r', 5), ('e', 6), ('a', 7), ('m', 8). Determine C, the total number of hash collisions that occur, and F, the current load factor after all the insertions. Assume the hash code for 'a' is 0, 'c' is 2, 'e' is 4, 'i' is 8, 'm' is 12, and 'r' is 17.

C: 3

F: $\frac{6}{8} = 0.75$

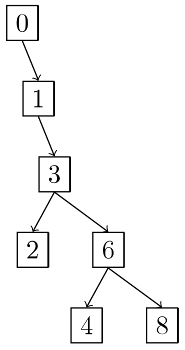
Count unique Keys only.



5 Dream of BSTification

Note: The tree question on the Thursday handout was different, as it uses a different LLRB invariant than what is discussed in this class ™.

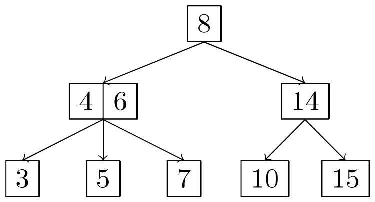
$\boxed{5.1}$ Let's say we are given an extremely unbalanced binary search tree:

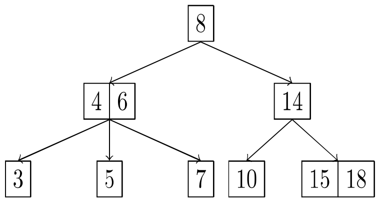


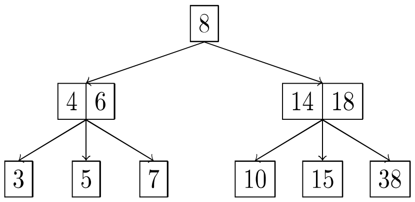
Write down a minimum length series of rotations (i.e. "rotate right/left on x") that will make tree balanced and have height of 2.

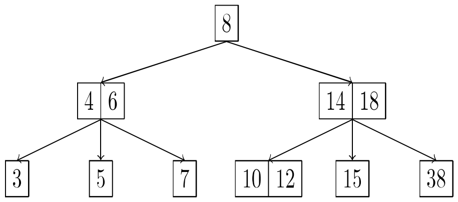
rotate left on 0
rotate left on 1

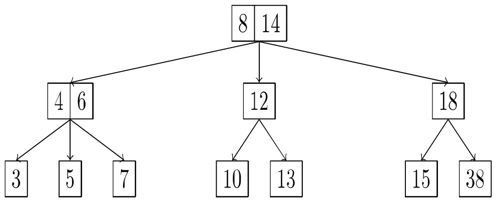
$\boxed{5.2}$ Draw what the following 2-3 tree would look like after inserting 18, 38, 12, 13, and 20.

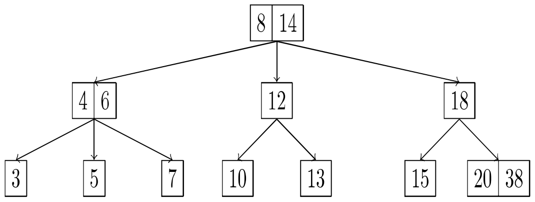




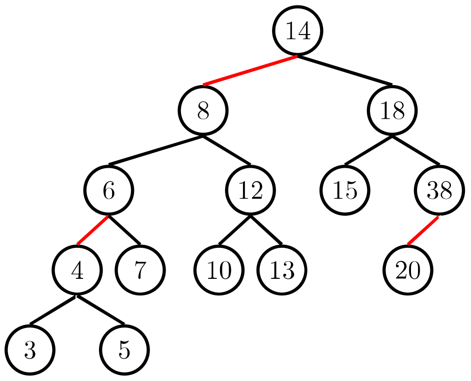








$\boxed{5.3}$ Now, convert the resulting 2-3 tree to a left-leaning red-black tree.


14
8
18
6
12
15
38
4
7
10
13
20
3
5