CSCB63 Assignment 1

Zheyuan Wei

February 12, 2023

Q1

Q1-a

Q: *If* $f(n) \in \mathcal{O}(g)$ *and* $g \in \mathcal{O}(h)$ *then* $f \in \mathcal{O}(h)$, *for all* f, g, h *in* $\mathbb{N} \to \mathbb{R}^+$

Proof.

1) From $f \in$ O(g) *we have* :

$$\exists c_1, n_1 \in \mathbb{R} \text{ s.t. } f(n) \leq c_1 \cdot g(n) \forall n \geq n_1$$

2) From $g \in$ O(h) *we have* :

$$\exists c_2, n_2 \in \mathbb{R} \text{ s.t. } g(n) \leq c_2 \cdot h(n) \forall n \geq n_2$$

3) From 1) and 2) we have:

$$f(x) = \begin{cases} f(n) \leq c_1 \cdot g(n) \forall n \geq n_0, \text{for some } c_1 \text{ and } n_1 \\ g(n) \leq c_2 \cdot h(n) \forall n \geq n_1, \text{for some } c_2 \text{ and } n_2 \end{cases} \Rightarrow f(n) \leq c_1(c_2 \cdot h(n)) \forall s.t. \begin{cases} n \geq n_1 \\ n \geq n_2 \end{cases}$$

4) From 3) we have:

$$f(n) \leq c_1 \cdot c_2 \cdot h(n) \ \forall n \geq max(n_1, n_2)$$

$$\Rightarrow f \in \mathcal{O}(h)$$

□

Q1-b

Q: *If* $f \in \Omega(g)$ *and* $g \in \Omega(h)$ *then* $f \in \Omega(h)$, *for all* f, g, h *in* $\mathbb{N} \to \mathbb{R}^+$

Proof.

1) From $f \in \Omega(g)$ we have:

$$\exists c_1, n_1 \in \mathbb{R} \text{ s.t. } f(n) \geq c_1 \cdot g(n) \forall n \geq n_1$$

2) From $g \in \Omega(h)$ we have:

$$\exists c_2, n_2 \in \mathbb{R} \text{ s.t. } g(n) \geq c_2 \cdot h(n) \forall n \geq n_2$$

3) From 1) and 2) we have:

$$f(x) = \begin{cases} f(n) \geq c_1 \cdot g(n) \forall n \geq n_0, \text{for some } c_1 \text{ and } n_1 \\ g(n) \geq c_2 \cdot h(n) \forall n \geq n_1, \text{for some } c_2 \text{ and } n_2 \end{cases} \Rightarrow f(n) \geq c_1(c_2 \cdot h(n)) \forall s.t. \begin{cases} n \geq n_0 \\ n \geq n_1 \end{cases}$$

4) From 3) we have:

$$f(n) \geq c_1 \cdot c_2 \cdot h(n) \ \forall n \geq max(n_0, n_1)$$

$$\Rightarrow f \in \Omega(h)$$

□

Q1-c

Q: $\log_\phi(\sqrt{5}(n+2)) - 2 \in \mathcal{O}(\log_2(n))$ where ϕ is the golden ratio.

Proof. Let $f(n) = \log_\phi(\sqrt{5}(n+2)) - 2, \forall n \geq 0$

Then it follows that:

$$f(n) = \log_\phi(\sqrt{5}(n+2)) - 2 = \frac{\log_2(\sqrt{5}(n+2))}{\log_2(\phi)} - 2$$

$$f(n) = \frac{\log_\phi \sqrt{5}}{\log_2(\phi)} \cdot \log_2(n+2) - 2$$

$$f(n) < \frac{\log_\phi \sqrt{5}}{\log_2(\phi)} \cdot \log_2(n+2) = k \cdot \log_2(n+2) \ \forall n \geq 0$$

where $k = \frac{\log_\phi \sqrt{5}}{\log_2(\phi)} \in \mathbb{R}$

$$\Rightarrow f(n) < k \cdot \log_2(n+2) \leq k \cdot \log_2(n^2) = k \cdot 2 \cdot \log_2(n) \ \forall n \geq 2$$

which is equivalent to:

$$f(n) < 2k \cdot \log_2(n) \ \forall n \geq 2$$

$$\Rightarrow f(n) \in \mathcal{O}(\log_2(n))$$

□

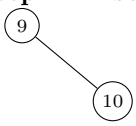
Q2

Q2-a

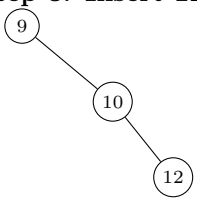
Q: On an initially empty tree, show each step of inserting the keys 9, 10, 12, 14, 3, 34, 19, 37, 20.

Step 1: Insert 9

Step 2: Insert 10



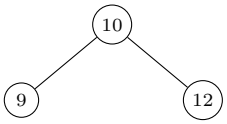
Step 3: Insert 12



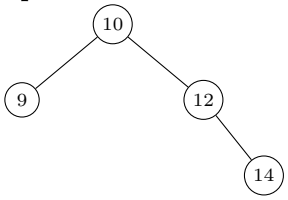
Imbalanced. Right heavy at node 9.

Need a left rotation.

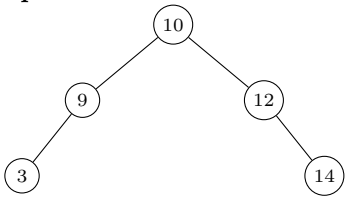
Left rotation:



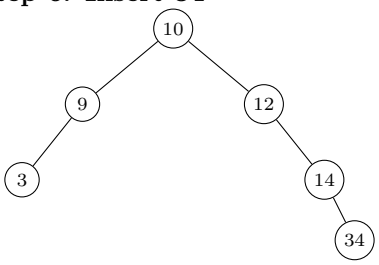
Step 4: Insert 14



Step 5: Insert 3



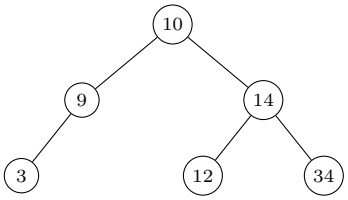
Step 6: Insert 34



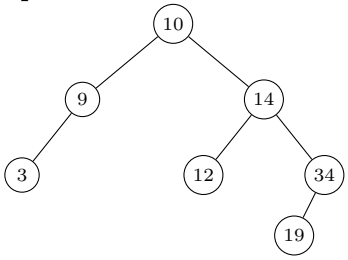
Imbalanced. Right heavy at node 12.

Need a left rotation.

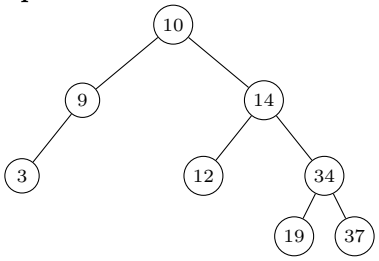
Left rotation:



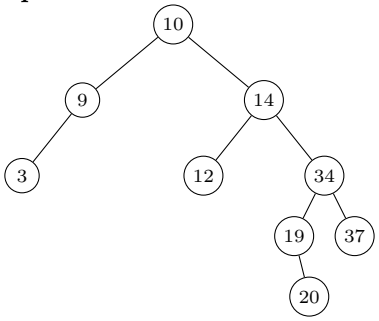
Step 7: Insert 19



Step 8: Insert 37



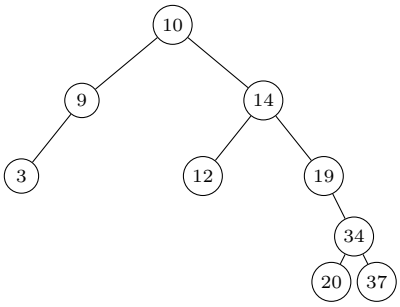
Step 9: Insert 20



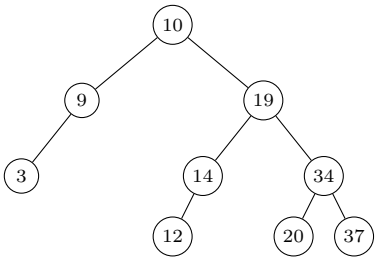
Imbalanced. Right heavy at node 14.

Need a double rotation.

1. Right rotation:



2. Left rotation:

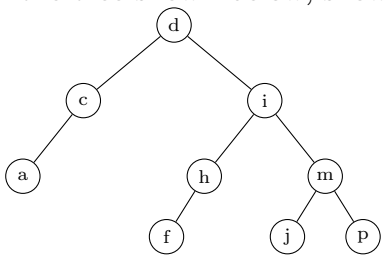


■

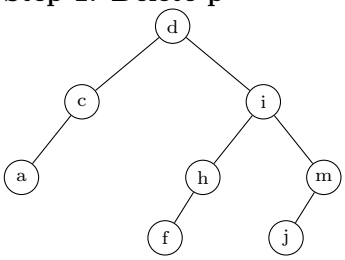
Q2-b-1

Q:

On the tree shown below, show each step of deleting the keys p, d, h.



Step 1: Delete p

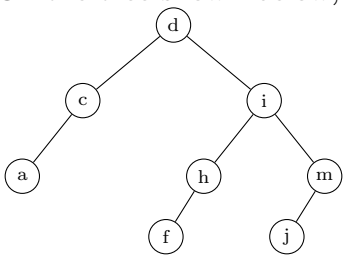


■

Q2-b-2

Q:

On the tree shown below, show each step of deleting the keys p, d, h.

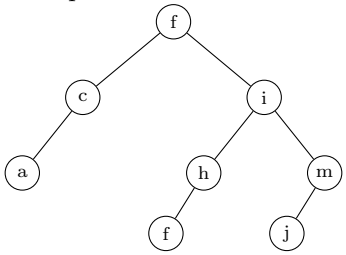


Step 2: Delete d

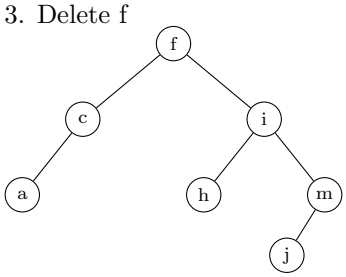
1. Find the successor of d:

   Which is the leftmost node in the right subtree of d, which is f.

2. Replace d with f



3. Delete f

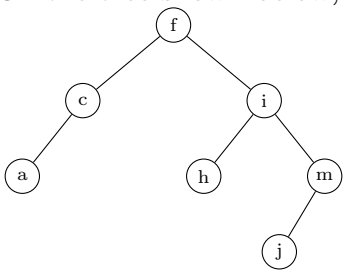


■

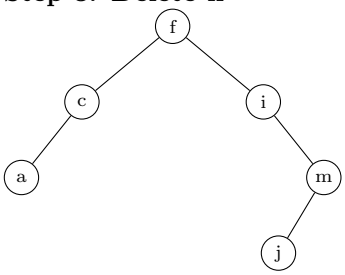
Q2-b-3

Q:

On the tree shown below, show each step of deleting the keys p, d, h.



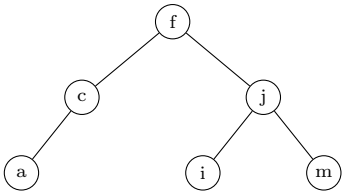
Step 3: Delete h



Imbalanced. Right heavy at node i.

Need a left rotation.

Left rotation:

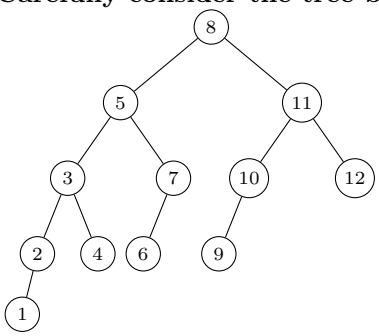


■

Q2-c

Q:

Carefully consider the tree below.



- If we start with an empty AVL tree, what sequence of insertions would result in this tree?

- Show each step of deleting key 11.

1. Insertion Sequence: (one possible sequence)

- Insert 8
- Insert 5
- Insert 11
- Insert 3
- Insert 7
- Insert 10
- Insert 12
- Insert 2
- Insert 4
- Insert 6
- Insert 9
- Insert 1

The only constraint is the key 1 must be inserted last. ■

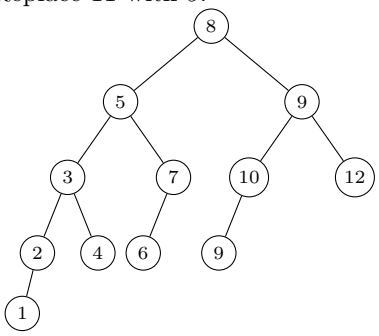
2. Delete key 11:

Step 1: Find the seccessor of 11.

The leftmost node in the right subtree of root, i.e. 9.

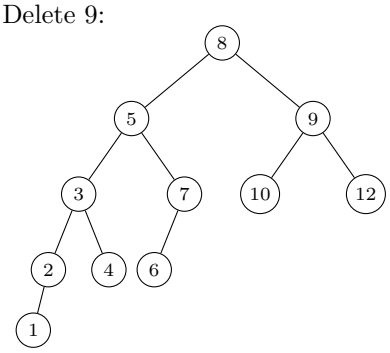
Step 2: Replace 11 with 9.

Replace 11 with 9:



Step 3: Delete 9.

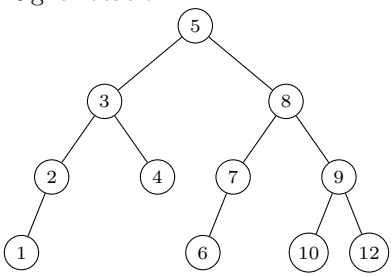
Delete 9:



Imbalanced. Left heavy at node 8.

Need a right rotation.

Right rotation:



■

Q4

Q: Consider an ADT consisting of a set S of distinct integers and the following operations:

- search(S, x): Return true if x is in S and false otherwise.
- insert(S, x): Insert the element x into the set S. This operation has no effect if x is already in S.
- delete(S, x): Delete the element x from the set S. This operation has no effect if x is not in S.
- min difference(S): Given a set S with size of at least 2, return a pair of distinct integers (x, y), with x $\in$ S, y $\in$ S, with the minimum absolute difference, i.e.
  $\forall x', \forall y', (x' \neq y' \rightarrow |x - y| \leq |x' - y'|)$

Your task is to design a data structure to implement this ADT,
such that all operations are performed in $\mathcal{O}(\log\ n)$ time,
where $n = |S|$. You will do so by augmenting our familiar AVL tree.

1. Describe all information that will be stored in the nodes.
2. Provide pseudo-code for each required operation.
3. Justify why your algorithms are correct and why they achieve the required time bound.

A Evil Desing

1. Describe all information that will be stored in the nodes.

Given: A set S of distinct integers.

Also, the number of input must be finite. (In practical)

Also, we only discuss the time complexity, not the space complexity.

Then we come up with an evil design:

Say we have a bound for the value of the input, then we create a boolean array of size n

we put each input into the array, and use True to represent the input is in the set, False to represent the input is not in the set, and we use the value of the input as the index of the array.

Then it follows that the search, insert, and delete are all $\mathcal{O}(1)$

-

■

1. Describe all information that will be stored in the nodes.

- int key;
- void *data;
- int height;
-

■