ShanghaiTech University

CS101 Algorithms and Data Structures
Fall 2024

Homework 5

Due date: November 6, 2024, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to Gradescope.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Gradescope account settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. We recommend you to write in LaTeX.
6. When submitting, match your solutions to the problems correctly.
7. No late submission will be accepted.
8. Violations to any of the above may result in zero points.

1. (12 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)	(e)	(f)

(a) (2') Which of the following statements about heaps are true?

A. In a complete binary min-heap, the sum of values of the internal nodes is no greater than that of leaf nodes.

B. We can build a heap from an array in linear time through Floyd's method, so we can sort an array in linear time through heap sort.

C. If we use a min-heap to do heap sort, we can sort an array in increasing order.

D. For any node a in a heap, the subtree of the tree with root a is still a heap.

Solution: C is controversial. You can get full points if you choose AD or ACD.

C.

(b) (2') Which of the following statements about the Huffman Coding Algorithm are true?

A. Huffman Coding Algorithm is a compression method without information loss.

B. If character a has a higher frequency than b, then the encoded a has a length no longer than encoded b.

C. The Huffman Coding Tree is a complete binary tree.

D. Given the set of characters and the order of their frequencies but the exact frequencies unknown, we can still determine the length of each encoded character.

(c) (2') Which of the following can be a set of decoded characters in the Huffman Coding Algorithm?

A. $\{0, 10, 110, 1110, 11110, 111110\}$

B. $\{00, 0100, 0101, 011, 10, 11\}$

C. $\{0, 100, 101, 10, 11\}$

D. $\{00, 010, 011, 110, 111\}$

(d) (2') Suppose there are two arrays: $\{a_i\}_{i=1}^n$ is an **ascending** array with n distinct elements. $\{b_i\}_{i=1}^n$ is the reverse of a, i.e. $b_i = a_{n-i+1}$. Which of the following statements is true?

A. If we run Floyd's method to build a min-heap for each of the two arrays, the resulting heap will be the same.

B. If we build a complete binary min-heap by inserting a_i sequentially into an empty heap, the runtime of this process is $\Theta(n)$.

C. If we build a complete binary min-heap by inserting b_i sequentially into an empty heap, the runtime of this process is $\Theta(n)$.

D. If we run heap sort on $\{b_i\}_{i=1}^n$, the runtime is $\Theta(n)$.

(e) (2') Which of the following statements about BST are true?

A. For a BST, the newly inserted node will always be a leaf node.

B. Given an array, suppose we construct a BST (without balancing) by sequentially inserting the elements of the array into an empty BST. Then the time complexity of this process is $O(n \log n)$ in all cases.

C. Given a BST with member variable `tree_size` (the number of descendants of this node) and a number x, we can find out how many elements in the BST is less than x in $O(h)$ time where h is the height of the BST.

D. If a BST is constructed by inserting elements in a random order, the expected height of the tree is $O(\log n)$.

(f) (2') Which of the following statements about BST are true?

A. In a BST, the nodes in a subtree appear contiguously in the in-order traversal sequence of the BST.

B. If a node doesn't have a right subtree, then its **next** (defined in lectures) object is the largest object (if any) that exists in the path from the node to the root.

C. In a BST, the pre-order traversal sequence can uniquely determine the structure of the tree.

D. In a BST, the sum of the heights of all nodes is minimized when the tree is perfectly balanced.

2. (6 points) Huffman Coding

After you compress a text file using the Huffman Coding Algorithm, you accidentally spill some ink on it and you find that one word becomes unrecognizable. Now, you need to recover that word given the following information:

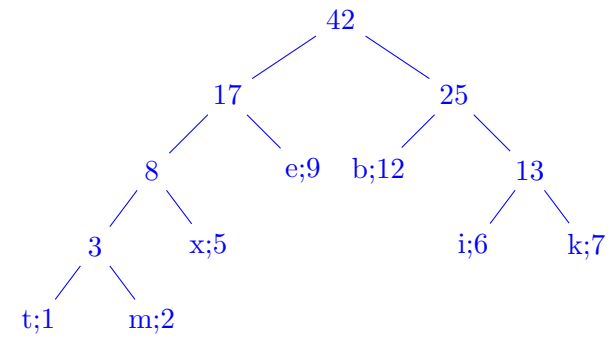
Huffman-Encoded sequence of that word:
00001100111101
Frequency table that stores the frequency of some characters:

characters	b	e	i	k	m	t	x
frequency	12	9	6	7	2	1	5

(a) (4') Please construct the binary Huffman Coding Tree according to the given frequency table and draw the final tree below.
Note: The initial priority queue is given as below. When popping nodes out of the priority queue, the nodes with the same frequency follows "First In First Out".

t	m	x	i	k	e	b
1	2	5	6	7	9	12

Solution:

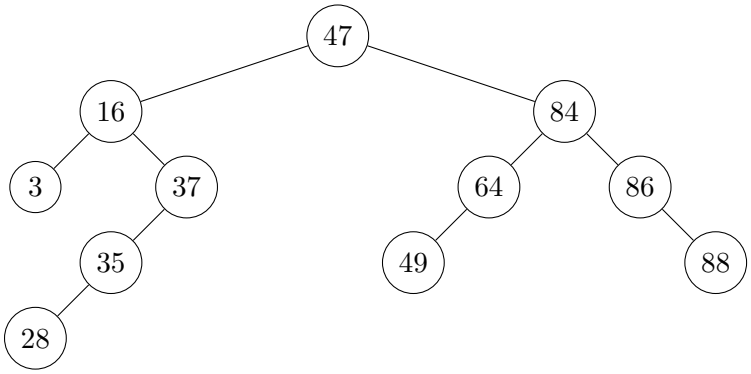


(b) (2') Now you can "decompress" the encoded sequence and recover the original word you lost. Please write the original word below.

Solution: tieke

3. (7 points) Binary Tree Practice

(a) (2') Consider the Binary Search Tree (BST) T shown below, which satisfies the BST property (where each node's key is the integer value itself) but is not height-balanced. Identify all nodes that are not height-balanced.



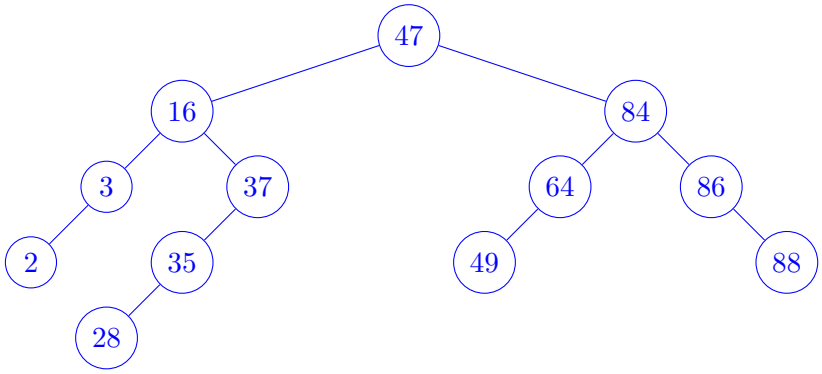
Solution: The nodes containing the keys 16 and 37 are not height balanced.

(b) (5') Perform the following insertions and deletions, one after another in sequence on T, by adding or removing a leaf while maintaining the binary search tree property (a key may need to be swapped down into a leaf). For this part, do not use rotations to balance the tree. Draw the modified tree after each operation:

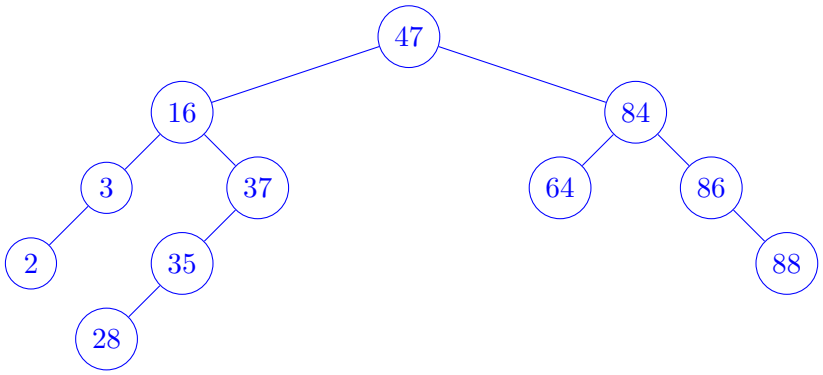
```
T.insert(2)
T.delete(49)
T.delete(35)
T.insert(85)
T.delete(84)
```

Solution:

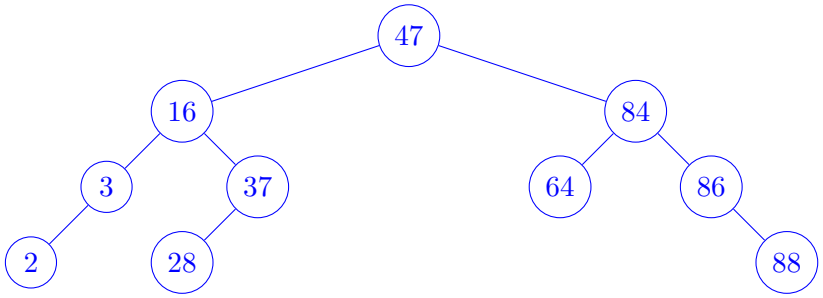
After Insert 2:



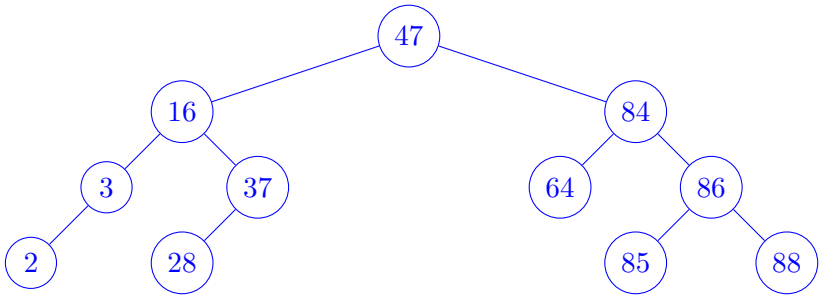
After Delete 49:



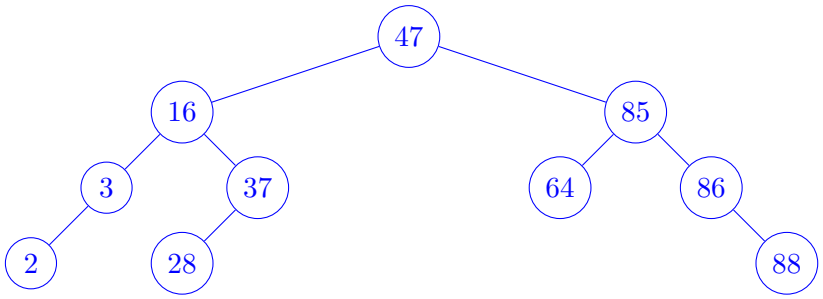
After Delete 35:



After Insert 85:



After Delete 84:



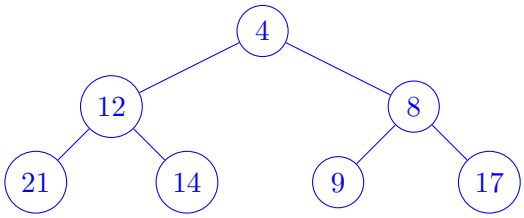
4. (8 points) Heap Practice

For each array provided:

1. Represent the array as a complete binary tree.
2. Determine if the tree is a max-heap, a min-heap, or neither.
3. If the tree is neither, convert it into a min-heap by repeatedly swapping adjacent items within the tree. Illustrate each swap by drawing the resulting sequence of trees, clearly indicating the swapped pairs on each tree.

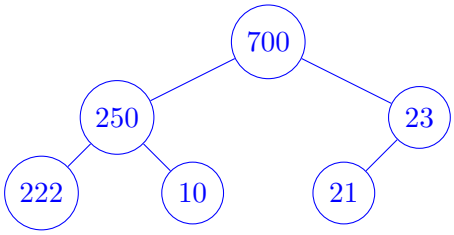
(a) 4, 12, 8, 21, 14, 9, 17

Solution: Min-heap.



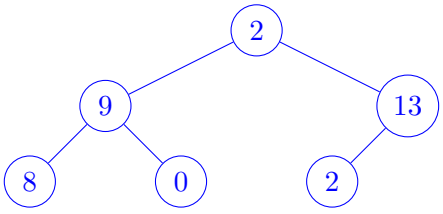
(b) 700, 250, 23, 222, 10, 21

Solution: Max-heap.

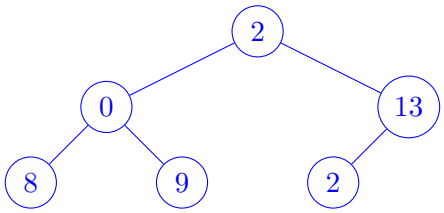


(c) 2, 9, 13, 8, 0, 2

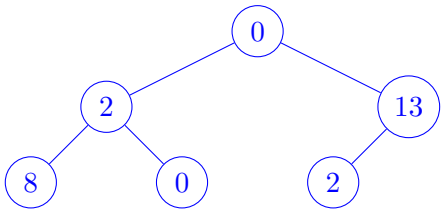
Solution: Neither: three swaps suffice to transform into a min-heap.



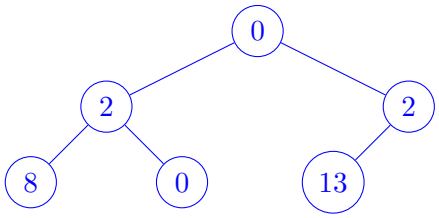
Swap 0 and 9:



Swap 2 and 0:



Swap 2 and 13:



5. (12 points) BST with Duplicates

In our lecture, we require each BST node stands for a single unique element. However, in this question we are talking about BST with duplicated elements. That is, we need to maintain how many identical elements are in the BST. Now, each node may stands for multiple elements with the same value, and we call it the `count` of a node. Here we assume the value is `int` type.

First we give the definition of our `Node` struct.

```
struct Node {
    int val;    // The value of the node.
    int sumCount;    // The number of all elements in the sub-tree
    Node *left, *right; //the left and right child.
};
```

Note that `sumCount` stands for the number of all elements in the **entire sub-tree**, i.e. the sum of `count` in the entire sub-tree. You will see why we use this definition in the following questions.

For example, if root r has two children a and b, and neither a nor b has a child. Suppose r.`count` $= n_r$, a.`count` $= n_a$ and b.`count` $= n_b$. Then r.`sumCount` $= n_r + n_a + n_b$, a.`sumCount` $= n_a$, and b.`sumCount` $= n_b$.

(a) (3') After figuring out the definition of member variable `sumCount`, you need to design a function that calculates the `count` of a node, using `sumCount` variable. Please complete this function below, and make sure you do not access `nullptr`.

```
// return how many elements a single node stands for
int get_count(Node *a){
    if(a == nullptr) return 0;
    int ans = /* (1) */ ;
    if(a->left != nullptr) ans = /* (2) */ ;
    if(a->right != nullptr) ans = /* (3) */ ;
    return ans;
}
```

Solution:

(1) `a->sumCount`

(2) `ans = ans - a->left->sumCount`

(3) `ans = ans - a->right->sumCount`

(b) (4') Given a value v, we want to figure out how many elements are no more than v. Complete the function below, and make sure you do not access `nullptr`. You may use `get_count` function if needed. By calling `count_no_more_than(root, v)`, we can get the number of such elements in the entire BST.

```
// return how many elements are no more than v.
int count_no_more_than(Node *a, int v){
    if(a == nullptr) return 0;
    int ans = 0, tmp = 0;
    if (a->left != nullptr)
        tmp = /* (1) */ ;
    if(v < a->value)
        ans = /* (2) */ ;
    else if(v > a->value)
        ans = /* (3) */ ;
    else
        ans = /* (4) */ ;
    return ans;
}
```

Solution:

(1) `a->left->sumCount`

(2) `count_no_more_than(a->left, v)`

(3) `tmp + get_count(a) + count_no_more_than(a->right, v)`

(4) `tmp + get_count(a)`

(c) (2') Now suppose you have finished the following two functions correctly. Given the root node of our BST, and two integers l and r, how do you find out the number of elements in the range $[l, r]$? Use variables `root`, `l`, and `r`, and function `count_no_more_than` to write an expression that computes the desired number.

```
count_in_l_to_r = /* Your code */ ;
```

Solution:

`count_no_more_than(root, r) - count_no_more_than(root, l-1)`

(d) (3') **True or False**

(i) If we insert an element into our BST with duplicates, we may need to modify multiple nodes. √ **True** ◯ False

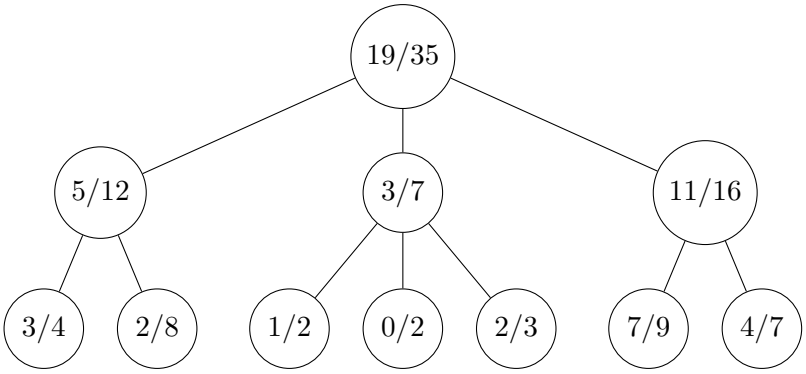
(ii) If we delete a node in our BST with duplicates, the nodes that we need to modify are the nodes on the path from this node to the root. ◯ True √ **False**

(iii) If we use member variable `count` instead of `sumCount`, we can still run `count_no_more_than` in $O(h)$ time, where h is the height of the tree. ◯ True √ **False**

6. (6 points) MCTS practice

In a Monte Carlo Tree, each node represents a state. In a node, using a/b to represent its historical information, which means this state won a times in b times of search. In this problem, we use $UCB = \frac{a_i}{b_i} + \sqrt{\frac{\log_2(b_p)}{b_i}}$, where node p is the parent node of node i.

Here is a part of a Monte Carlo Tree, containing the first three layers (i.e. the fourth and deeper layers are omitted).



(a) (2') Given a Monte Carlo Tree above, which node in the third layer will be selected according to the MCTS algorithm?

A. 3/4
B. 2/8
C. 1/2
D. 0/2
E. 2/3
F. 7/9
G. 4/7

(b) (4') After selecting a leaf node (deeper than the third layer), in the simulation step, the algorithm performs 5 simulations and wins 3 times. Please draw the new Monte Carlo Tree (the first three layers) after the Backpropagation step.

Solution: