Rodger Lee

CS32

UID: 204914719

Homework #5

1a.

A close up of a piece of paper

Description automatically generated

1b.

Pre-order: 50, 20, 10, 15, 40, 30, 25, 34, 60, 70, 65, 85, 76

In-order: 10, 15, 20, 25, 30, 34, 40, 50, 60, 65, 70, 76, 85

Post-order: 15, 10, 25, 34, 30, 40, 20, 65, 76, 85, 70, 60, 50

Level-order: 50, 20, 60, 10, 40, 70, 15, 30, 65, 86, 25, 34, 76

1c.

After deleting 30:

A flock of birds

Description automatically generated

After deleting 20:

A close up of a document

Description automatically generated

2a.

struct Node

{

Node(int myVal)

{

value = myVal;

left = right = parent = nullptr;

}

int value;

Node \*left, \*right, \*parent;

};

2b.

Input: A value v to insert

If the tree is empty

Allocate a new node and put V into it (with left = right = parent = nullptr)

Point the root pointer to our new node

Start at the root of the tree

While we haven’t reached the end of the tree

If V is equal to the current node’s value

Exit the function without doing anything (the value already exists)

If V is less than the current node’s value

If there’s a left child

Go left (set the current node equal to its left pointer)

Else

Allocate a new node and put V into it. Set current node’s left pointer to the new node. Set the new node’s parent pointer to the current node. Exit the function.

If V is greater than the current node’s value

If there’s a right child

Go right (set the current node equal to its right pointer)

Else

Allocate a new node and put V into it. Set current node’s right pointer to the new node. Set the new node’s parent pointer to the current node. Exit the function.

3.

Consider the following operations on an initially empty heap h; this heap is a **maxheap**, so the biggest item is at the top. The heap is represented as a binary tree:

h.insert(3);

h.insert(5);

h.insert(2);

h.insert(0);

h.insert(10);

h.insert(4);

int item;

h.remove(item); // Removes the biggest item from the heap, and puts it in item

h.insert(9);

h.insert(7);

h.remove(item);

a,b)

A close up of text on a white surface

Description automatically generated

c)

A close up of text on a white background

Description automatically generated

4.

1. vector<pair<string, list<int>>>, where each pair in the outer vector represents a course and all the students in that class, with those students being sorted in order. The pairs are in no particular order in the outer vector. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?

**O(C + S).** In the worst case, it will take C steps to find the class in the outer vector, and S steps to find the name of a particular student S.

1. map<string, list<int>>, where the students in each list are in no particular order. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?

**O(logC + S).** The map is ordered so a binary search will yield a big-O of logC. Searching an item through the list, ordered or unordered will take a maximum of S steps.

1. map<string, set<int>>. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?

**O(logC + logS).** Finding a particular item in a map and a set both yield big-O of logN.

1. unordered\_map<string, set<int>>. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?

**O(logS).** An unordered map is organized as a hash table, so on average the time complexity is O(1).

1. unordered\_map<string, unordered\_set<int>>. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?

**O(1).** Both containers are organized as hash tables.

1. Suppose we have the data structure map<string, set<int>> and we wish for a particular course *c* to write the id numbers of *all* the students in that course in sorted order. What is the big-O complexity?

**O(logC + S).** Binary search through map to find the class, and we must visit each student S in the set to print their id numbers.

1. Suppose we have the data structure unordered\_map<string, unordered\_set<int>> and we wish for a particular course *c* to write the id numbers of *all* the students in that course in sorted order (perhaps using an additional container to help with that). What is the big-O complexity?

**O(SlogS).** To find that particular course we must take 1 step. To write the numbers in sorted order, there is another set that contains the students in a sorted fashion. It takes S steps to visit each student in the unordered\_set, and logS steps to find that student in the other sorted container.

1. Suppose we have the data structure unordered\_map<string, set<int>> and we wish for a particular student *s* to write *all* the courses that student is enrolled in, in no particular order. What is the big-O complexity?

**O(ClogS).** This time complexity is the steps it takes to find the student, who may possibly be in every single class. We need to check every class, (C steps), and find the student in that class (logS steps).