CptS 223 - Advanced Data Structures in C++
Fall 2020

Take-Home Exam 2

Friday, November 13, 2020

Your Name: Koji Natsuhara

ID Number: 11666900

TA's Name: Nathan Wisner

READ THE FOLLOWING INSTRUCTIONS:

This exam is take-home. You must work **individually** on this exam. You may use your book, notes, and online resources if necessary. Please either handwrite or type your answers into this document. Late exam solutions will **not** be accepted! **Please show all work!!!**

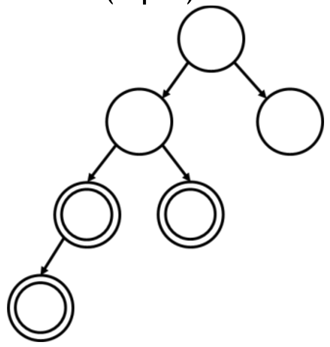
You must submit the exam through Git by Monday, November 16, midnight. On your local file system, and inside of your Git repo for the class, create a new branch called Exam2. In the current working directory, also create a new directory called Exam2. Convert your exam solution file to a.pdf called Exam2.pdf. The Exam2.pdf file must be added and committed. Merge your Exam2 branch with your Master branch and push the Master to the remote origin. Be sure to answer each question precisely. Do not provide superfluous details in your answers. *NOTE: you do not need to comment your code solutions.*

Section	**Points Possible**	**Points Earned**
1	25	
2	15	
3	40	
4	20	
Total	**100**	

Your Name: ___________________________ TA's Name: ___________________________

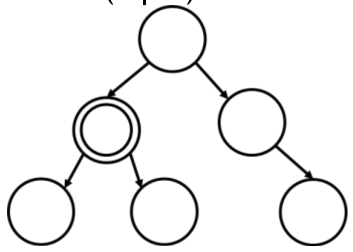
Section 1: Red-black Trees

1. (5 pts) Is the following tree a valid red-black tree? Why or why not?



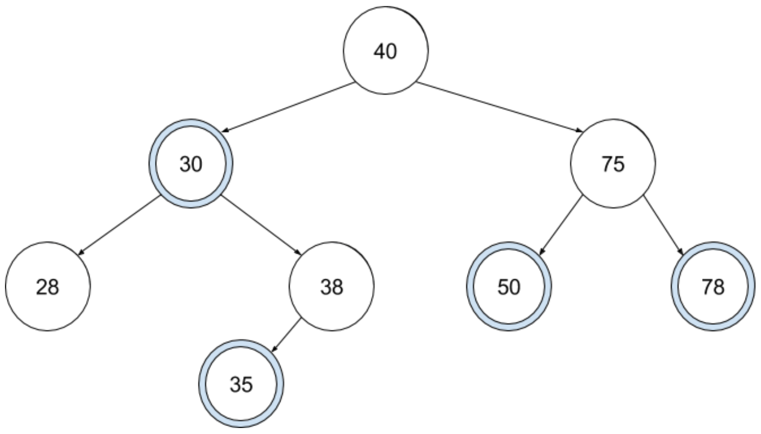
The following tree is not a valid red-black tree because a red node cannot have a red child node.

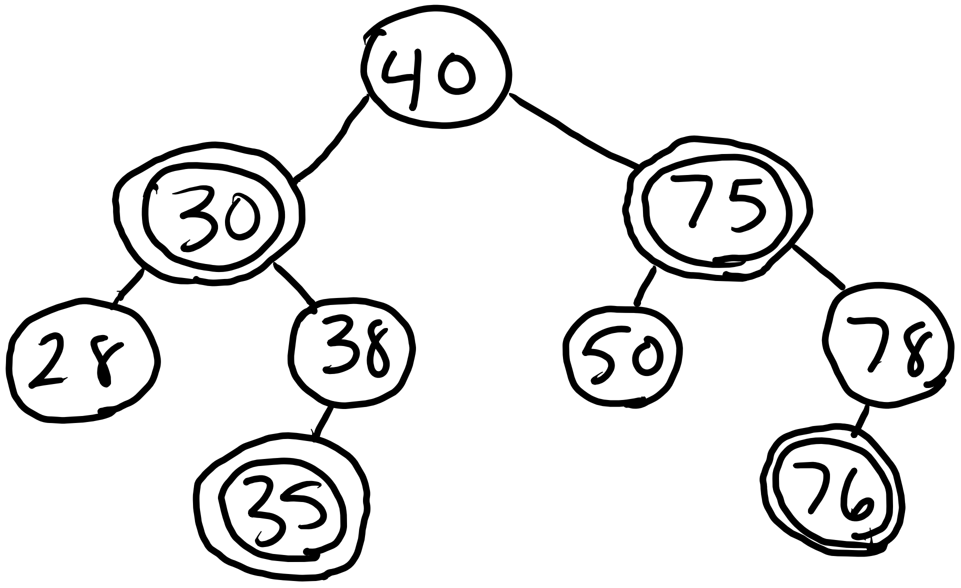
2. (5 pts) Is the following tree a valid red-black tree? Why or why not?



The following tree is not a valid red-black tree because the height of the black nodes from the root are not equal. The left subtree has a height of 1 black node while the right subtree has a height of 2 black nodes.

3. (5 pts) Insert 76 into this Top Down Red Black tree:





4. (5 pts) When deleting a node from a red-black tree, we always want the node we remove to be colored _________ [red/black/either]. Explain your answer.

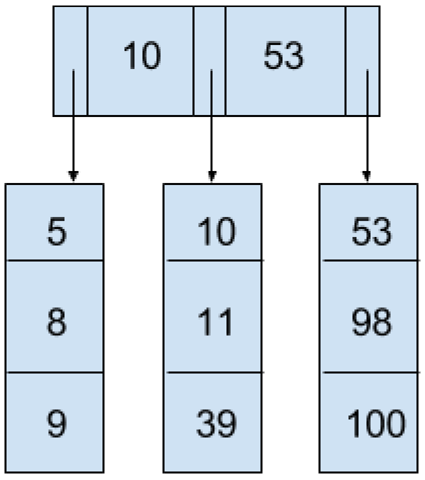
When deleting a node from a red-black tree, we always want the node we remove to be colored red because removing a red node will have little to no effect on the red-black tree coloring system. If the node that needs to be deleted is colored black, then it is guaranteed to violate the rule that all pathways must have the same number of black nodes.

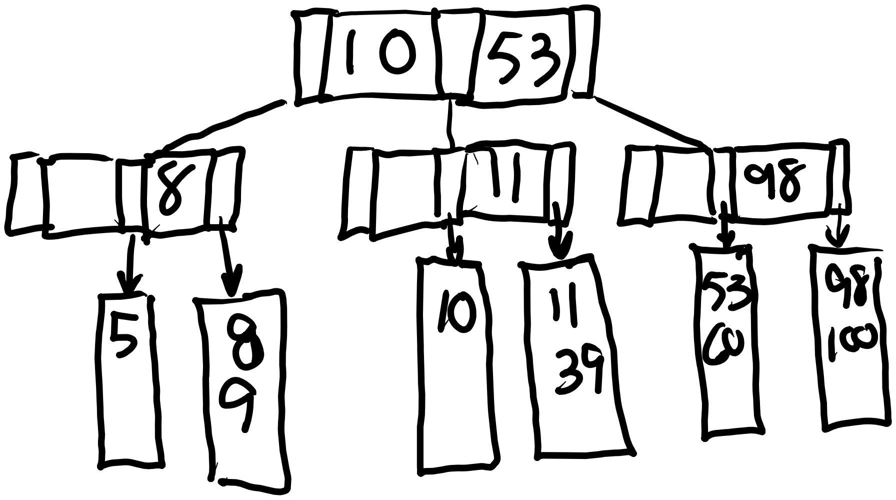
5. (5 pts) AVL vs. Red-Black trees: These are both balanced trees, but why would we use one over the other?

AVL and Red-Black trees are both self-balancing trees and have a runtime complexity of O(logn), but Red-black trees are preferred over AVL trees because Red-black tree rules are less strict than AVL trees. Because of this, less memory and operations are done in Red-black tree insertions without needing to rotate all the time like AVL tree insertions.

Section 2: B+ Trees

6. (10 pts) The following B+ tree has M = 3, L = 3. Insert 60 into the tree and draw the results. Make sure your final tree is a valid B+ tree.





7. (5 pts) What is the primary reason to use a B+ Tree?

The primary reason to use B+ Trees is to store many records in a data structure without having to perform many operations to keep the tree balanced. Because it is easy to maintain a B+ tree's balance, it makes it easier to manipulate the data.

Your Name: ___________________________ TA's Name: ___________________________

Section 3: Hash Tables

8. (5 pts) List the key factors that affect the Big-O performance of a hash table:

The key factors that affect the Big-O performance of a hash table is how well the hash function sorts, the method in which the hash table handles collisions, and the initial table size.

9. (10 pts – 5 pts/table) Given a hashing function hash(x) = ((x * x) + x) % 11. Insert the value **17** into each hash table using the rules specified below. Note that some of the boxes in each hash table are already full. If there's a collision, start the probing at i = 0 and increment from there.

Linear Probing having probe(i) = (hash(x) + (i + 1)) % 11

10	11	17				2			6	15
0	1	2	3	4	5	6	7	8	9	10

Quadratic Probing having probe(i) = (hash(x) + (i^2 + i) + 1) % 11

10	11		17			2			6	15
0	1	2	3	4	5	6	7	8	9	10

10. (5 pts) Define the term "load factor" as it relates to hash tables.

The load factor of hash tables is the (total number of elements) / (the number of buckets) in the hash table. Ideally, you want to have load factor of below 0.5 before you have to rehash the entire table.

11. (5 pts) After we insert 17 into these hash tables, should we resize and rehash the values? Use Table Size and N to calculate the load factor (λ) and apply the general rule of when you rehash for linear and quadratic probing.

After we insert 17 into these hash tables, we should resize and rehash the values. The load factor of both of the tables is a load factor of 0.55 which is over the general threshold of a load factor of 0.5.

12. (5 pts) If this was a hash table using separate chaining instead of linear probing, would it be time to rehash after we insert 17? Explain.

It would not be necessary to rehash after you insert 17 if the hash table used separate chaining. Typically, you want to rehash when the load factor exceeds 1. In this case, the load factor of the hash table is still 0.55.

13. (5 pts) What is lazy deletion and why is it important for linear and quadratic probing?

Lazy deletion is when you do not actually delete the data that is meant to be removed, you just mark it as "deleted". Lazy deletion is important for linear and quadratic probing because by marking a slot as "deleted" or "empty" can help the search function from stopping prematurely. If an element is removed from the table and has caused collisions beforehand, then the hash function to search for an element will no longer work.

14. (5 pts) Which of the following hashing algorithms is likely to produce a faster HashTable implementation? Why?

Hashing Algorithm A	Hashing Algorithm B

int hash(string text):
  int result = 0;
  for(char ch : text)
    result += ch;
  return result;
	int hash(string text):
	  int result = 1;
	  for(char ch : text)
	    result *= ch * ch + (11 * ch) + 7;
	  return result;

Hashing Algorithm B is more likely to produce a faster HashTable implementation because of the amount of variance of integers that the strings will produce. Because of this, the strings inserted into the hash table will be less likely to collide with each other.

Your Name: _________________________ TA's Name: _________________________

Section 4: Parallel Programming

15. (5 pts) What is parallel programming? Explain.

Parallel programming is the usage of multiple cores/processors to perform a set of partitioned various tasks in a program so that they can run in coordination to speed up the runtime of the program/operations. Parallel programming is mainly used to decrease the runtime of a program.

16. (5 pts) When should we use critical regions in OpenMP? Explain.

We use critical regions in OpenMP when you want threads to perform an operation one at a time.

17. (5 pts) What is meant by synchronizing tasks? Explain.

Synchronizing tasks makes sure that each core works at the same speed so that one core does not get too far ahead of the rest. This imposes constraints and is used to protect access to shared data.

18. (5 pts) How can we apply parallel programming to hash tables? Explain.

We could apply parallel programming to hash tables by having the search function be performed in parallel and so that it can be performed faster.