CS260 - Data Structures
Written Assignment 2

Instructor: Yusuf Osmanlıoğlu

Due: March 10th, Thursday 2021, 11:59PM - No late submission

1 Instructions

This assignment is worth 100 points.

Submission:

- Write your answers in a pdf file using LaTeX. Use the provided answerSheet.tex file as a reference.
- Put your name at the top of the file. Name the generated pdf file with your student ID as **abc123_wa1.pdf** and submit it via Gradescope.
- When submitting your assignment, tag the questions in your answer sheet properly in Gradescope. 10 points will be subtracted from your total grade for improper question tagging.

2 Questions

Question 1 : 15 points

It's just another day in your internship at Google. You feel like life starting to get mundane as you are thirsty of a new challenge. What you were looking for arrives sooner than you expect, from an unexpected direction though: Binary Search Trees. Answer the following questions on BSTs:

(a) [5 points] Bob is another intern working in the same group as you. He was assigned the job of designing and implementing a binary search tree (BST), which he did in a very short amount of time, becoming the favourite intern of the group leader. However, you have a gut feeling that, BST's that Bob's code generates do not always satisfy the BST property. Not out of jealousy, of course, but for your good will towards your employer, you mentioned your concerns to the group leader. She assigned you the job of developing an algorithm, which will test whether a given BST is valid or not.

Suggest an algorithm in plain English that tests whether a given binary tree is a valid binary search tree. Also do the runtime analysis of your algorithm.

(b) [5 points] Realizing your algorithm design skills, your group leader decided to test your boundaries. Although she is now convinced that you know something about BSTs (at least more than Bob), she wants to make sure that you understand the matter and do not "memorize" these data structures and algorithms. To further test the depth of your understanding in BST, she is now asking you to develop an algorithm that would determine whether two sets of keys that represent two BSTs are the same BSTs, without building the BSTs.

Provide an example for two sets of keys $A = a_1, a_2, ..., a_k$ and $A' = a_1, a_5, a_7 ..., a_k$, that are different in order but having the same set of elements, yet generate the same BST if inserted in order. You can assume that the elements will be integers.

(c) [5 points] As a follow up to part b above, suggest an algorithm in plain English and/or pseudocode that determines whether the two sets represent the same BST. Also do the runtime analysis of your algorithm.

Question 2 : 15 points

Your project leader has been quite impressed by your performance on binary search trees, and started entertaining the idea of keeping you full time after you graduate. However, she needs to make a case to her manager about how magnificent a problem solver you are. Thus, she decided to put a few more challenges for you. This time she wants to test how well versed you are with heaps. You are given an unsorted array of size n. Below are the set of questions that you need to solve to prove your worth by using this unsorted array.

(a) [2 points] The naive way of generating a heap with that array is by using a separate list of the same size and generating the heap by inserting elements one by one. Give a $O(n \log n)$ time algorithm to generate the heap.

(b) [8 points] That was easy. How about generating the heap in place? Give pseudocode or explain in plain English of a function, that generates a heap without using a second list. You can only use upheap/downheap functions that we used in class, while other functions (such as insert/deleteMin) are not allowed. What is the runtime of your algorithm?

(c) [5 points] Although we stated that generating the heap would take $O(n \log n)$ above, it was actually a loose bound which assumes that each insertion will take $O(\log n)$ time, for each of the n insertions. However, as you might have noticed when going though your iterations in the previous part of the problem, fixing the heap property for some nodes (or with the similar idea, inserting elements to a heap in early stages) do not always take $O(\log n)$ where, n is the size of the final heap. In fact, earlier stages takes less time, revealing that our initial $O(n \log n)$ time might have been a loose bound. Make a careful analysis of the make heap algorithm and show that it has $O(n)$ time.

Question 3 : 15 points

Below is a set of questions that would test the boundaries of your red-black trees understanding.

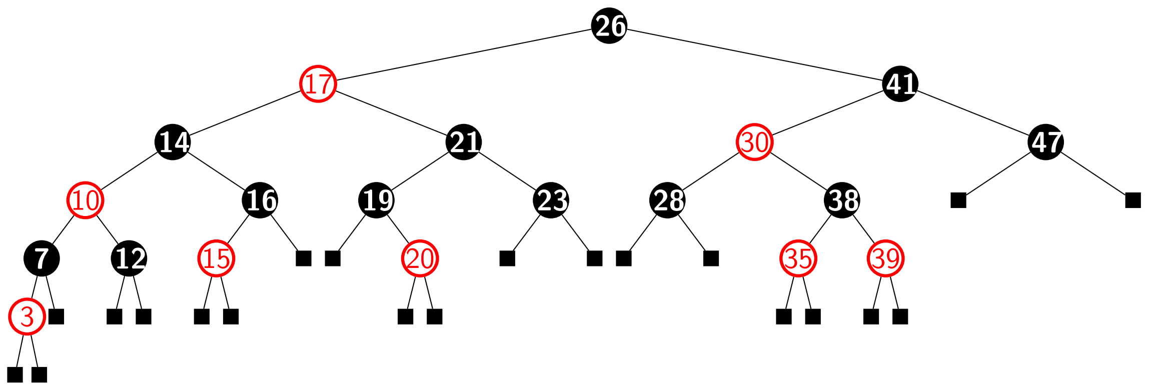
A valid Red-Black tree should satisfy the following 5 requirements:

1. Every node is either red or black.
2. The root is black.
3. Every leaf is NULL and black.
4. If a node is red, then both its children are black.
5. All paths from a node x to any leaf have same number of black nodes in between (i.e., their Black-Height(x) is the same)

After insertion of a new key x to the tree, properties 3 and 5 can get violated, which requires re-balancing of the tree. Following are the possible cases of violation for the red-black properties:

1. x's parent is the left child of x's grandparent while x's uncle is Red.
2. x's parent is the left child of x's grandparent, x's uncle is Black, and x is right child of its parent.
3. x's parent is the left child of x's grandparent, x's uncle is Black, and x is the left child of its parent.
4. x's parent is the right child of x's grandparent while x's uncle is Red.
5. x's parent is the right child of x's grandparent, x's uncle is Black, and x is right child of its parent.
6. x's parent is the right child of x's grandparent, x's uncle is Black, and x is the left child of its parent.

These violations can be overcome by rotating the tree around certain nodes and making updates in their colors, as described in the slides. First three of these cases along with how to handle them were described in the slides, from which you can deduce how to handle the rest, as the latter three are the mirroring cases of the former three.



(a) [5 points] Considering the Red-Black tree given above, insert the key 36 to this tree. Which cases are you going to encounter? What rotations would you need to overcome these cases? What would the final tree look like? (Note: It suffices to show the final state in picture, but explain verbally what rotations you will do to achieve to that state due to which violations)

(b) [5 points] What is the asymptotic cost of inserting a new key to a red-black tree? Explain your answer.

(c) [5 points] Where would you use red-black trees? Why?

Question 4 : 15 points

Answer the following questions about graphs.

(a) [3 points] What is the maximum number of edges that can exist in an undirected graph? Why? Explain.

(b) [3 points] What is a cycle in a directed graph?

(c) [3 points] Given a graph $G(V, E)$, its inverse consists of a graph G' where the edges of the original graph is inverted. That is, for all $(u, v) \in G.E$, $\exists(v, u) \in G'.E$. Describe a nonempty directed graph G such that $G = G'$.

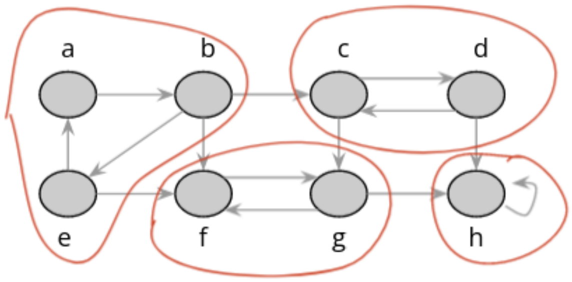
(d) [3 points] Topological sorting in a directed graph is a linear ordering of its vertices. Does existence of a cycle in a graph violate a topological sorting? Why/why not?

(e) [3 points] Depth First Search can be used as a subroutine to solve other problems, topological sorting being an example of this. Explain the intuition behind using DSF in the calculation of topological sort. (Note: I'm not asking you about how DFS is used in topological sorting. Rather, I'm asking what is needed to topologically sort graphs and how DFS becomes useful to achieve this)

Question 5 : 15 points

A *strongly connected component* (SCC) of a directed graph $G(V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u and v in C, we have both $u \rightsquigarrow v$ and $v \rightsquigarrow u$; that is, vertices u and v are reachable from each other.

In the example graph given below, the sets of nodes a, b, e, c, d, f, g, and h each form a strongly connected component. Thus, the graph consists of 4 strongly connected components.



(a) [7 points] Breadth First Search (BFS) can be used in determining whether each node in a given directed graph is reachable by every other node. Give the pseudocode of an algorithm using BFS that determines whether a given directed graph is a single SCC. Also provide the explanation of your algorithm in plain English along with a runtime analysis.

(b) [8 points] Depth First Search (DFS) can be used to calculate strongly connected components in a graph. Give the pseudocode of an algorithm using DFS that calculates SCCs in a given directed graph. Also provide the explanation of your algorithm in plain English along with a runtime analysis.

Question 6 : 10 points

You are given a weighted undirected graph $G = (V, E)$, where E and V denote set of edges and vertices, and a minimum spanning tree T of that graph G. Answer the following questions about G and T on minimum spanning trees.

(a) [5 points] Suppose we decrease the weight of one of the edges in G that is not among the edges in T. Suggest an algorithm in plain English that determines whether T is still a minimum spanning tree, and if it is not, calculates a minimum spanning tree of G. Explain the running time of your algorithm. (Note: Your algorithm should be faster than Prim's and Kruskal's)

(b) [5 points] Consider the following algorithm running on $G = (V, E)$. Would it calculate a valid MST of G? If yes, explain your reasoning in plain English, if no, provide a counter example graph that the algorithm will fail producing an MST. Also, what is the running time of the algorithm in the previous question. Show your analysis.

```
MSTcandidate(G)
 E = sort G.E in decreasing order of edge weights
 T = E
 for i from 1 to |E| do
     if T - {E[i]} is a connected graph
         T = T - {E[i]}
     end if
 end for
 return T
```

Question 7 : 15 points

In economics, arbitrage is the practice of taking advantage of a difference in prices in two or more markets; striking a combination of matching deals to capitalize on the difference, the profit being the difference between the market prices at which the unit is traded.

Arbitrage can become a source of gain when applied to currency exchanges. Consider the following currency exchange ratios as an example: 1 U.S. dollar buys 0.7292 euros, 1 euro buys 105.374 Japanese yen, 1 Japanese yen buys 0.3931 Russian rubles, 1 Russian ruble buys 0.0341 U.S. dollars.

Then you could take 1 U.S. dollar and buy 0.7292 euros with it, with which you can buy 76.8387 yen (because 0.7292 * 105.374 = 76.8387). Then take the 76.8387 yen and buy 30.2053 rubles (because 76.8387 * 0.3931 = 30.2053), and finally take the 30.2053 rubles and buy 1.03 dollars (because 30.2053 * 0.0341 = 1.0300).

Assume that you are given the pairwise trading ratios among n currencies and answer the following questions about arbitrage accordingly.

(a) [5 points] Describe the problem as a graph problem. What are your nodes? What are the edges? Is this a weighted graph? Is the graph directed? What is the goal over this graph?

(b) [5 points] Suggest an algorithm in plain English that evaluate whether an arbitrage exists among these n currencies.

(c) [5 points] Expand your algorithm so that it prints the chain of arbitrage and analyze the running time of your algorithm.