CS29003 ALGORITHMS LABORATORY
(Class Test II)
Date: 21 – November – 2020

Question 1

Give the visited node order for each type of graph search, starting with s, given the following adjacency lists and the accompanying figure (Figure 1):
$adj(s) = [a; c; d]$,
$adj(a) = []$,
$adj(c) = [e; b]$,
$adj(b) = [d]$,
$adj(d) = [c]$,
$adj(e) = [s]$.

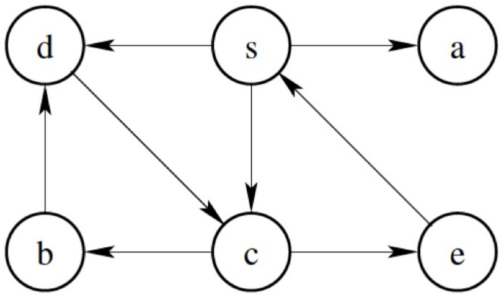


Figure 1: Example graph.

Graph search techniques: (a) BFS, (b) DFS.
Solution:
BFS: s a c d e b
DFS: s a c e b d
Marking scheme: [2.5+2.5], Boolean marking, no part marks.

Question 2

Answer **true** or **false** with **short justifications**.

(a) While running DFS on a directed graph, if from vertex u we visit a finished vertex v, then the edge $(u; v)$ is a cross-edge.
Solution: False.
Explain: The edge could be either a cross-edge or a forward edge. Note: The edge cannot be a back edge—a back edge goes to a vertex that has started, but not finished.

(b) Let P be a shortest path from some vertex s to some other vertex t in a directed graph. If the weight of each edge in the graph is increased by one, P will still be a shortest path from s to t.
Solution: False.
Explain: Counterexample: See figure 2.

(c) Dynamic programming is more closely related to BFS than it is to DFS.
Solution: False.

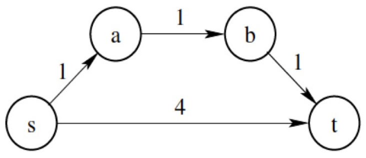


Figure 2: Counterexample.

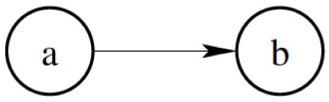


Figure 3: Example.

Explain: DFS is more closely related. The top-down approach to dynamic programming is effectively performing DFS on the subproblem dependence graph. The bottom-up approach means solving subproblems in the order of a reverse topological sort, which is also related to DFS.

(d) A depth-first search of a directed graph always produces the same number of tree edges (i.e., independent of the order in which the vertices are provided and independent of the order of the adjacency lists).
Solution: False.
Explain: The DFS forest may contain different numbers of trees (and tree edges) depending on the starting vertex and upon the order in which vertices are searched. Consider the example in figure 3. If the DFS starts at a, then it will visit b next, and (a, b) will become a tree edge. But if the DFS visits b first, then a and b become separate trees in the DFS forest, and (a, b) becomes a cross edge.

(e) Suppose we do a DFS on a directed graph G. If we remove all of the back edges found, the resulting graph is now acyclic.
Solution: True.
Explain: If DFS finds no back edges, then the graph is acyclic. Removing any back edges found doesn't change the traversal order of DFS, so running DFS again on the modified graph would produce no back edges.

Marking scheme: [0.5+1.5 each. No part marking.]

Question 3

You are given a list of all scheduled daily flights in India, giving departure airports, departure times, destination airports, and arrival times. We want an algorithm to compute travel times between airports, including waiting times between connections. Assume that if one flight arrives at time t and another departs at time $t' \geq t$, travelers can make the connection. Further assume that at a given airport, no two events (arrivals or departures) occur at the same time, and that there are at most 100 events at any airport during a given day. All times are given in GMT and India has a single time zone. We can construct a weighted graph so that given a departure airport, a departure time T, and a destination airport, we can efficiently determine the earliest time T' that a traveler can be guaranteed (according to the schedules!) of arriving at her destination on that day (ignore overnight flights and overnight airport stays).

We create a vertex for every flight departure and every flight arrival. That is, each vertex corresponds to a particular time (arrival or departure of a flight) at a particular airport. We also create one vertex for the departure time at the departure airport. We create an edge for every flight. The weight of these edges is the flight time. We also create an edge from every event (departure or arrival) at an airport to the next event at the same airport. The weight of these is the time between the events.

- Give an upper bound on the number of edges and vertices in your graph if there are n airports in India and m daily flights. Justify your bound.
  Solution: For the construction given above, the number of vertices is clearly $2m$. The number of edges is a bit harder to bound. Here it is $99n + m$, since there are at most 100 events at an airport (so 99 intervals between them) and m flight edges. One can also obtain a bound of $3m - 1$.

- State true or false with one line justification: The graph construction time at best can be $O(V \log V)$ where V is the number of vertices in the graph.
  Solution: Use BFS, DFS, Dijkstra or even Fibonacci heaps. Once the graph is constructed, we can find the shortest travel times using BFS or DFS; since our vertices are associated with a time and an airport, all the reachable vertices from the starting vertex represent times in which a traveler can be at an airport. By finding the earliest reachable vertex per airport we find the solution.

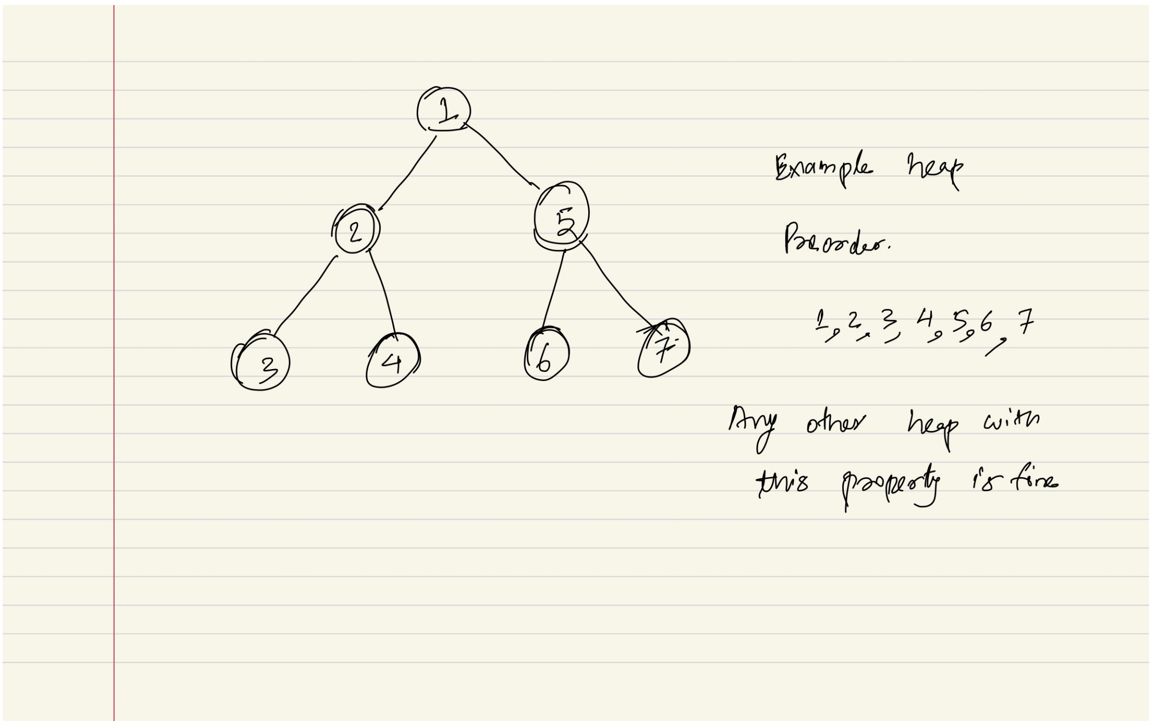
Marks: Part a: nodes – 1.5 marks (no part marking), edges – 2.5 marks (no part marking.)
b. Boolean marks, no part marking.

Question 4

Answer **true** or **false** with **short justifications**.

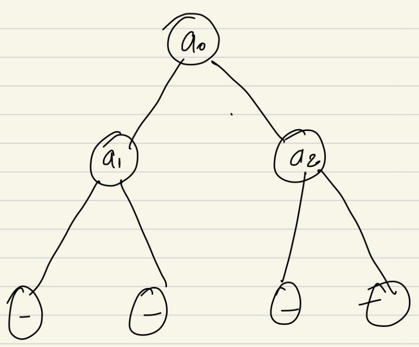
(a) There is a heap T storing seven distinct elements such that a *preorder* traversal of T yields the elements of T in sorted order. If your answer is **true**, draw the heap. If your answer is **false**, give a brief justification.

Solution: True



(b) The same problem as above but this time with an *inorder* traversal.

Solution: False



Inorder will list

$a_1 \ldots < a_0 \ldots < a_2$

for heap property

either $a_0 > \{a_1, a_2\}$

or $a_0 < \{a_1, a_2\}$

Hence it's not possible.

(c) We discussed the median of medians method to select a pivot, that provides a worst-case linear-time algorithm for the k^{th} order statistics. If instead of a group of 5 elements, we take a group of 3 elements, we still get a worst-case linear time algorithm. You may justify your answer using the final recurrence relation, do not need to show its derivation.

Solution: False

Justification: The recurrence relation (approximate) $T(n) \leq T(2n/3) + T(n/3) + O(n)$ Which on solving gives $T(n) = O(nlogn)$

(d) For priority queue implementation, we need three main operations: deleteMax, findMax and insert. There is an algorithm that can perform each of these operations in $O(1)$. If your answer is **true**, provide the algorithm. Otherwise, explain briefly why this is not possible.

Solution: False

Justification: If it was possible, you could come up with a sorting algorithm in comparison model, that works in $O(n)$. Given n elements, insert them into a priority queue one by one – $O(n)$ time. Then perform $deleteMax$ n times – $O(n)$ time again. This gives you the sorted sequence in $O(n)$ time. This is not possible as per our lower bound on comparison based sorting.

Marking scheme: [0.5+1.5 each. No part marking.]

[8 marks]

Question 5

Suppose you have a book shelf, that can only be accessed from left to right. Suppose you keep n books in the shelf. For each book b_i, you have a probability $p_i > 0$ with which you access, and it has a width w_i. Suppose, the books are placed in an order $b_1, b_2, \ldots, b_n$. To access the book b_i, you need an access time proportional to $\sum_{j=1}^{i} w_j$. Thus, you can define the expected access time for the books as

$$\sum_{i=1}^{n} p_i \sum_{j=1}^{i} w_j$$

You need to design a greedy algorithm to minimize the expected access time. Below, you are given two greedy algorithms.

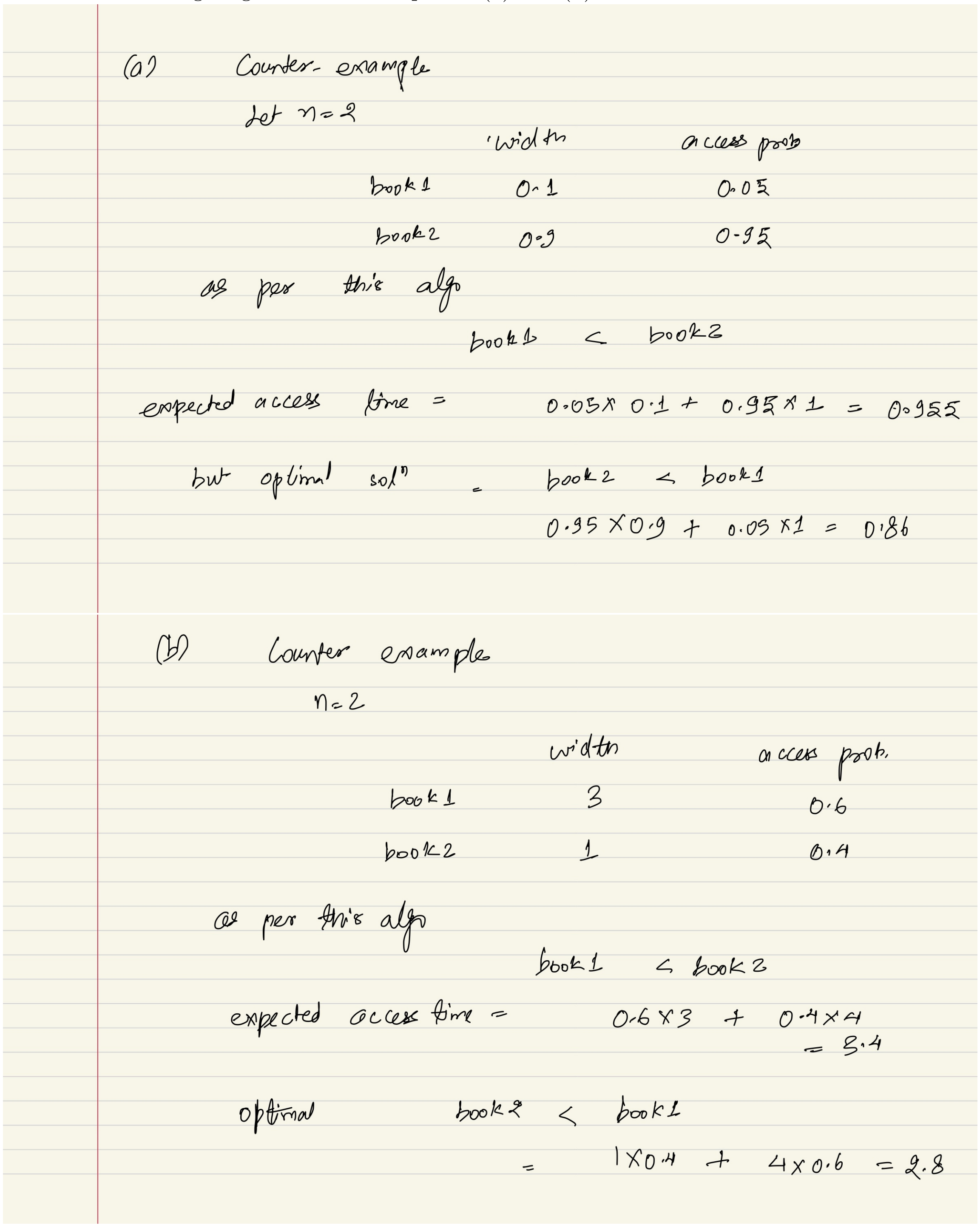
(a) Order the books from smallest to largest (as per width) on the shelf. That is, $w_i < w_j$ implies that $i < j$.

(b) Order the books from most likely to be accessed to least likely to be accessed. That is, $p_i < p_j$ implies that $i > j$.

For each of the greedy algorithms, if the algorithm is correct, mention that. If it is incorrect, provide a counter-example. If both the algorithms are incorrect, you need to provide counter-examples for both, and also mention the correct greedy algorithm. No proofs are required.

[7 marks]

Solution: Both (a) and (b) are not correct. (1 mark each for telling that these are incorrect).
1.5 marks each for giving a counter-example for (a) and (b). See below.



(a) Counter-example

Let $n = 2$

	width	access prob
book1	0.1	0.05
book2	0.9	0.95

as per this algo

book1 < book2

expected access time = $0.05 \times 0.1 + 0.95 \times 1 = 0.955$

but optimal sol^n = book2 < book1

$0.95 \times 0.9 + 0.05 \times 1 = 0.86$

(b) Counter example

$n = 2$

	width	access prob.
book1	3	0.6
book2	1	0.4

as per this algo

book1 < book2

expected access time = $0.6 \times 3 + 0.4 \times 4$
$= 3.4$

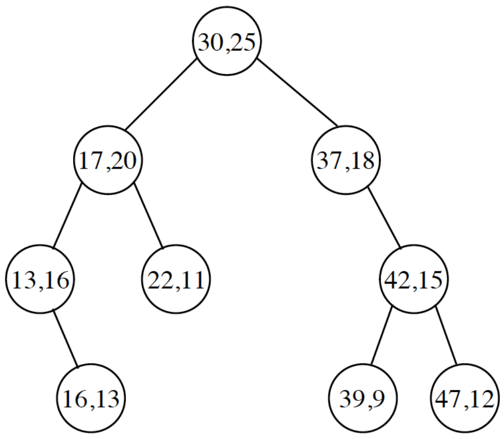
optimal book2 < book1

$= 1 \times 0.4 + 4 \times 0.6 = 2.8$

2 marks for the *correct algorithm:* Order the books from the smallest ratio of width over access probability to the largest ratio. That is, $w_i/p_i < w_j/p_j$ implies that $i < j$.

Question 6

Consider the following diagram. Each node stores a pair (x, y), with x denoting a value and y denoting its priority. See that while the values follow binary search tree property, the priority scores follow a heap property (not the heap structure).



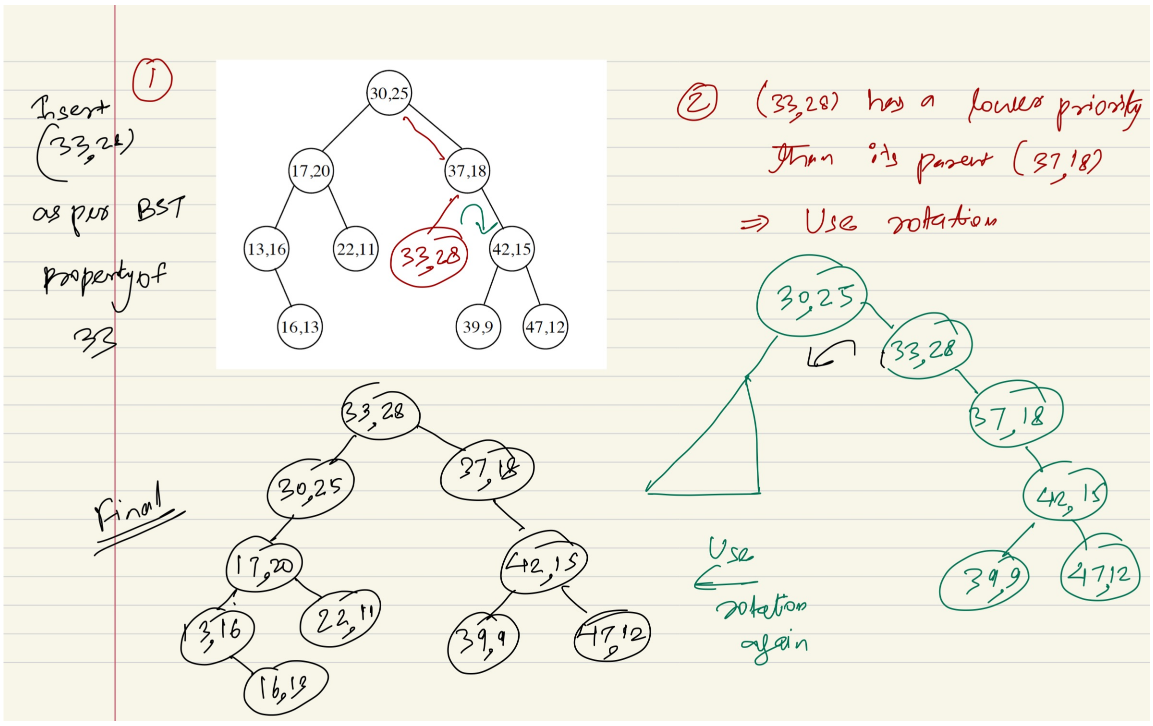
- Briefly explain how you can insert a new item (a, b) into this structure so as to follow the above mentioned restrictions.

  Solution: Insert as per the correct BST ordering of a. It will be inserted as a leaf node. Now, check if heap property is being violated because of b. If so, use rotations to traverse upto the root. If it is a left child, use right rotation on the parent. If it a right child, use left rotation. BST property does not change on rotations. [3 marks for talking about both rotations. Deduct one mark if the answer mentions to just use rotations after inserting as per BST ordering. Otherwise, 0 marks]

  Alternative solution: One can also use a top-down approach – First we descend in the tree (as in a regular binary search tree by a), and stop at the first node in which the priority value is less than b. We have found the place where we will insert the new element. Next, we call Split (T, a) on the subtree starting at the found node, and use returned subtrees L and R as left and right children of the new node.

- Show the modified structure after inserting $(33, 28)$.

  Solution: 3 marks for showing the final structure. **Intermediate structures not needed.** If final structure not correct, give 1 mark if one of the intermediate structures is correct.



[6 marks]