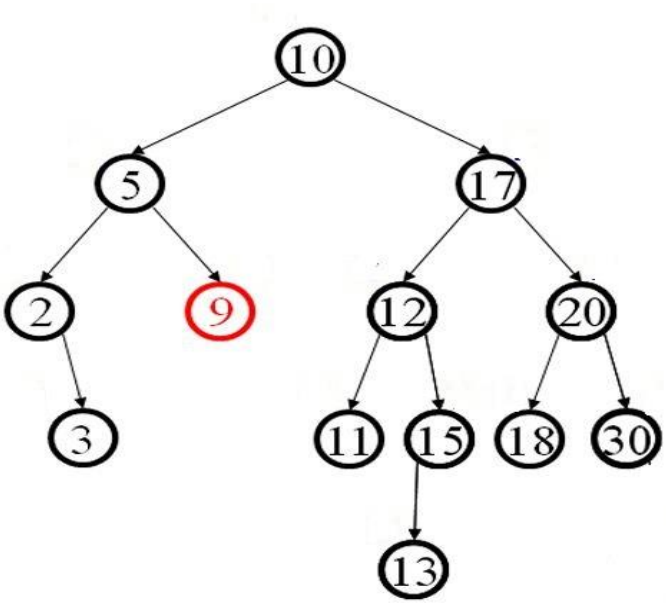
B

Name: _____

ADA Final Exam

Short theoretical questions [3.5 points]

[Q 1.] [1 point]

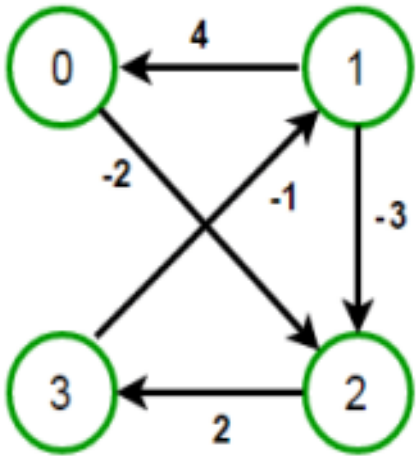


a.) What is an **AVL-Tree**? Formally define ***the concept*** of AVL Tree by explaining the restrictions that apply for its structure. What is its main ***usage and purpose***? [0.25 points]

b.) For the AVL tree in the figure, illustrate and **explain** how **insert** works, use the examples of inserting, in order, **14** and **16** [0.40 points]

c.) Having as starting point the same *AVL tree given in the **initial figure***, illustrate and **explain** how **delete** works, use the example of deleting **9** from the initially given tree [0.35 points]

[Q 2.] [0.75 point]



a) Formally define ***the concept*** of **shortest paths in a directed weighted** graph. [0.25 points]

b) Explain **Bellman-Ford algorithm** and illustrate how it works on the graph in the figure, having as source the **node 0**. Clearly explain which **datastructures** are needed by the algorithm and **show their content at every step of the algorithm**. You can chose some textual notation, or you can chose to re-draw the graph with some annotations at every step, just make sure that the steps of the algorithm are clearly visible and explained. [0.5 point]

[Q 3.] [0.75 point]

Briefly explain, based on examples, the concept of union-find (disjoint sets) with a forest of up-trees, with union by rank and path compression. Explain the datastructures needed to represent them. DO NOT WRITE CODE FOR IMPLEMENTING THE OPERATIONS, but DO A TRACE ON THE CONTENT OF THE DATASTRUCTURES for the following example:

a.) Initially, we have following disjoint sets: [A, B, C] [D, E, F, G] [M, N] [R, S, T]. For this initial configuration, depict a possible situation of *trees* representing these disjoint sets and also show the ***content* of the datastructures** used to represent them.

b.) The following sequence of operations is performed: Union (A, M); Union (A, R); Find(S). Depict the resulting configuration after each one of these operations: after every operation in this sequence, draw the trees and also show the content of the datastructures.

[Q 4.] [1 point] Identify, explain and correct the problems in the following code.

a.) Make the changes to the code in order to correct the problem [0.25 point]

b.) Clearly and in detail **explain** why it was a problem and what changes by your correction. Give concrete examples of the incorrect results produced by the function before the correction, and what they become after the correction. [0.75 point]

The code to be analysed is the method `put` given below. This method is part of a class `StringBST`. This class represent binary search trees having nodes with String keys. There are no associated values, only keys. Only a relevant excerpt is given below, parts of the implementation that are not relevant for this question are omitted.

```
class Node {
   // …
   // …  the usual node of a BST with String keys
 }

public class StringBST {
    private Node root;

    // …  omitted what is irrelevant for this purpose

   public void put(String key) {
        put(root, key);
    }
   private void put(Node x, String key) {
        if (x == null) x = new Node(key);
        int cmp = key.compareTo(root.key);
        if (cmp <0) put(x.left, key);
        else if (cmp >0)  put(x.right, key);
    }
   …

}
```

Algorithm design and implementation exercises [5.5 points]

Notes:

- *For all implementation exercises, you are **NOT required to write complete programs**! Focus **only** on implementing **what is required** - **specific operations** and the necessary data structures and variables.*
- *NO POINTS will be awarded if you implement some random operations that are not required!*
- *When appropriate, you should define and use **private helper functions or additional datastructures** to support the implementation of the required operations.*
- *For each of the 3 implemented operations, analyse its **time complexity***
- ***Add comments (free text and/or figures)** to explain and justify your implementation choices. These comments are meant to accompany your implementation code, and cannot serve as a replacement solution instead of the implementation code.*

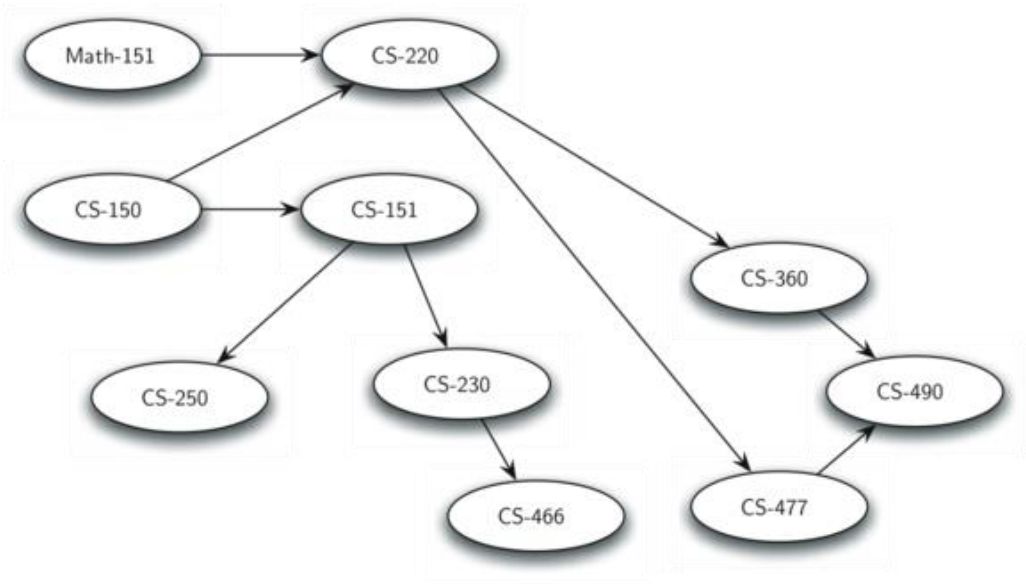
Graphs:

a.) Define the necessary data structures to represent a **graph of curricular prerequisites** where nodes are courses and edges indicate **prerequisite relationships** (i.e., an edge from course A to course B means A is a prerequisite for B, a student should first learn A and only then proceed to learn B). Courses are identified by their code (a string). The graph data structures must be based on adjacency **matrix**. Define the class `CourseGraph`, showing their class attributes and simple constructors. You do **not** have to implement a method to build the graph! [0.5 point]

b.) Make a drawing to show the contents of all the datastructures defined at a.) for the graph in the figure. [0.5 point]

c.) In the class CourseGraph, implement the method `isValid`, which returns true if this graph represents a valid curricula, and false otherwise (A curricula is valid if there are **no cyclic dependencies**, no course exists that transitively eventually depends on itself). [1 point]

d.) In the class CourseGraph, implement the method `getCourseOrder` which returns a list of all courses in a **good order**, such that **each course appears in the list after all of its prerequisites**. Chose an ***efficient*** implementation for `getCourseOrder`. [2 points]



Huffman trees:

a.) Define a HuffmanTree data structure. Define classes `Node` and `HuffmanTree`, showing their class attributes and simple constructors. There is no need to show the implementation of any other method (**no** need to build the Huffman Tree !). [0.5 point]

b.) In the `HuffmanTree` class, define a method `isCode` that takes as argument a String and returns true if it represents a valid code of some character from the tree, and false otherwise. [1 point]