Data Structures and Algorithms

XMUT-COMP 103 - 2026 T1

Mock Test

Agatha Rachmat

School of Engineering and Computer Science

Victoria University of Wellington

Admin

Assignment 6 is out

Due date:

20 June

Mock Test

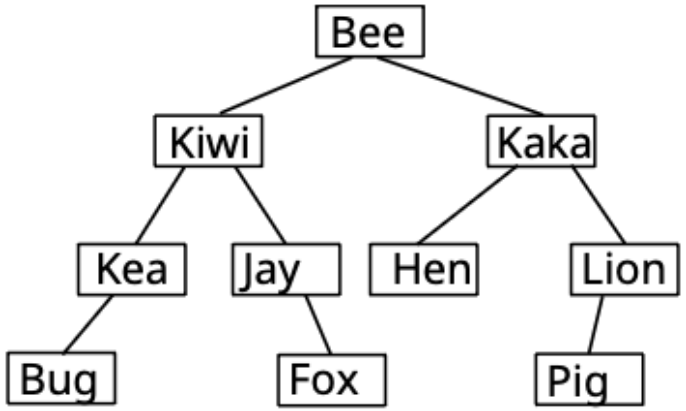
Try to do it yourself

Note:

There are only **pages 1-12**

Question 1. Tree Traversal Orders

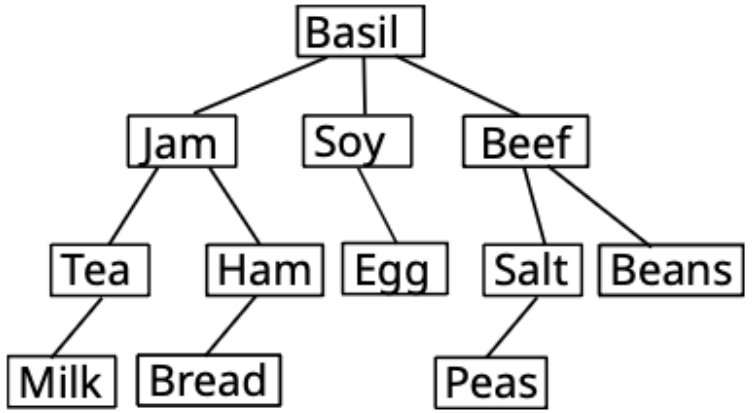
(a) [2 marks] For the binary tree below, give the order the nodes would be printed in if they were printed via a pre-order depth-first traversal.



Bee, Kiwi, Kea, Bug, Jay, Fox, Kaka, Hen, Lion, Pig

Question 1. Tree Traversal Orders

(b) [3 marks] For the general tree below, give the order the nodes would be printed in if they were printed via post-order depth-first traversal, assume children are processed left



Milk, Tea, Bread, Ham, Jam, Egg, Soy, Peas, Salt, Beans, Beef, Basil

Question 2. Binary Trees

(a) [5 marks] Complete the following printPostOrder method to print all the labels in a Binary Tree of LabelNodes in an post-order depth-first order. Hint: printPostOrder should be recursive.

```
public void printPostOrder(LabelNode node) {
      if (node != null) {
      printPostOrder(node.getLeft ());
      printPostOrder(node.getRight ());
          UI. print (node.getLabel (),"  ");
      }
}
```

Question 2. Binary Trees

(b) labels: F B G

Line with error: 3

because it assumes that both node.getLeft() and node.getRight() are not null.

Question 2. Binary Trees

(c) addLabels method which will add a Set of labels to a Binary tree of LabelNodes. For each label in the newLabels Set, the method makes a LabelNode containing the label and should then follow a random path down the tree from the root, until it reaches a node that has a null child (left, or right, or both). It should then add the new LabelNode as a child of that node.

- choose the left child if (Math.random()<0.5),
- and choose the right child otherwise.

assume that the root is not null.

```
public void addLabels(LabelNode root, Set<String> newLabels){
      for ( String label : newLabels){
            LabelNode newNode = new LabelNode(label); // new node to add
            LabelNode node = root;
            while (node.getLeft()!=null && node.getRight()!= null){
               if (Math.random()<0.5){node = node.getLeft();
               }
               else {node = node.getRight();
               }
            }
            if (node.getLeft()==null){node.setLeft(newNode); }
            else {node.setRight(newNode);}
```

Question 3. General Trees

(a)

```
public Person findPerson(Person person, String name){
    if (person==null || person.getName().equals(name)){
        return person;
    }
    for (Person employee : person.getEmployees()){
        Person p = findPerson(employee, name);
        if (p != null) { return p; }
     }
     return null ;
}
```

Question 3. General Trees

(b)

```
public void changeManager(Person person, Person newManager){
    Person oldManager = person.getManager();
    newManager.addEmployee(person);
    oldManager.removeEmployee(person);
    person.setManager(newManager);
}
```

Question 4. Traverse Graph 1

(a) return a Set of the Computers that are directly or indirectly connected to infectedComputer

```
public Set<Computer> findVulnerableComputers(Computer infectedComputer){
    Set<Computer> connectedComputers = new HashSet<Computer>();
    findVulnerableComputers(infectedComputer, connectedComputers);
    return connectedComputers;
}

public void findVulnerableComputers(Computer comp, Set<Computer> visited){
    visited .add(comp);
    for (Computer connected : comp){
        if (! visited.contains(connected) ){
            findVulnerableComputers(connected, visited );
        }
    }
}
```

Question 4. Traverse Graph 1

(b) returns true if a Computer is connected to an infected computer, directly or indirectly, in the network.

```
public boolean potentialThreat (Computer suspectedComputer, Set<Computer>
visited){
    if ( visited . contains(suspectedComputer)) return false;
    if (suspectedComputer.isDetected()){
       return true;
    }
     visited .add(suspectedComputer);
     for(Computer connected : suspectedComputer) {
        if ( potentialThreat (connected, visited )) {
           return true;
        }
     }
     return false ;
}
```

Question 5. Traverse Graph 2

```
public boolean isConnectedToXMUT(BusStop a, Set<BusStop> visited){
   if ( a. equals(XMUT) ) { return true; }
   visited .add(a);
   int count = 1;
   for (BusStop neighbour : a){
      if (! visited . contains(neighbour)) {
         if (isConnectedToXMUT(neighbour, visited)){
            return true;
         }
      }
    }
   return false ;
}
```