CS1010X — Programming Methodology
National University of Singapore

Practical Examination

Time allowed: 2 hours

Instructions (please read carefully):

1. This is **an open-book exam** that you are to attempt with your own laptop. Except for the laptop, you are not allowed to use other electronic devices inclusive of a mobile phone.
2. You are to do this assessment without any assistance from and communication with another human being (inclusive of, but not limited to, sharing of documents through cloud platforms or forums). You are not allowed to search for online help (inclusive of, but not limited to, google, stackoverflow, geeksforgeeks, chat-GPT, GPT4, co-pilot or equivalent for any development environments) outside of coursemology.org. If found otherwise, you will receive **ZERO** for this assessment and will be subjected to disciplinary actions.
3. This practical exam consists of **5** "topics" (with 7 questions or parts) printed in **9** pages (inclusive of this cover page).
4. The maximum score of this quiz is **32 marks** and you will need to answer all questions to achieve the maximum score. You are advised to use your time wisely and not being stuck on any question for too long.
5. Your answers should be submitted to coursemology.org **BEFORE** the end of the exam. If you have any submissions timestamped with a time after the exam has ended, your submission for that question will not be graded. Remember to **finalize** your submissions before the end of the exam. Also, you are to upload your screen recording (as previously communicated to you) of this assessment as done by you to Canvas by 23:59 on the day of the exam; otherwise, you will receive **ZERO** for this assessment.
6. You will be allowed to run some public test cases to verify that your submission is correct. Note that you can run the test cases on coursemology.org up to a **maximum of 15 times** for each question. Note that public test cases provided have explicit input and output stated.
7. You are also provided with some templates to work with - but we would like you to use them at your own discretion. If coursemology.org fails, you need to rename your file `practical_<mat_no>.py` where `<mat_no>` is your matriculation number and submit that file to Canvas. If in the worst case when the whole network is down, we will make arrangement there and then.
8. Please behave like a good programmer to make your codes readable with good naming convention for the variables and functions. Also, please keep your solution simple where applicable. If you do not do so (as you provided complex, complicated, or difficult to understand solutions), we reserve the right to deduct some credit from your total score.

Topic 1: Jumping Around to Locate [6 marks]

We have learned about binary search on a sorted list in Lecture 9. For this question, we (you, actually) want to experiment with a novel searching algorithm on a sorted list. Let's call it `jump_locate`. Here is an overview on how it works for a given `num_to_find` in a sorted list called `aList`, from `begin` (which is 0 at the beginning) till `end` (which is `len(aList)-1` at the beginning). It has a counter called `jump` that is initialized to 1, and has its value doubled sometimes when the algorithm has yet to reach the `num_to_find` (see Situation 2 below on when the doubling happens). So the function is

```
jump_locate(aList, begin, end, jump, num_to_find)
```

The algorithm starts with checking the very first element (`begin=0`) of `aList` to see whether it is the `num_to_find`. If yes, it is done and return the value of the index where the number is found (which is equal to `begin` in the very first test). If no, it checks `aList[begin+jump]` for 3 situations:

1. If `aList[begin+jump]` is equal to `num_to_find`, then we are again done and just return the answer `begin+jump`.
2. If `aList[begin+jump]` is less than `num_to_find`, then the index of the number to be found (if exists) should be between `begin+jump+1` and `end`, and we can recursively call the `jump_locate` function to find the number in this range. To speed up the searching process, in this case, we double the value of `jump`.
3. If `aList[begin+jump]` is greater than `num_to_find`, then the index of the number to be found (if exists) should be between `begin+1` and `begin+jump-1`, and we can also solve the problem by recursion. Since the length of the interval is less than `jump`, we should reset the value of `jump` to 1 for the recursion.

Notice that in the above discussion, we ignore the fact that `begin+jump` can possibly be larger than `end` since sometimes `jump` keeps doubling. This should be checked in your code and you should adjust the value of `jump` so that `begin+jump` is equal to `end` for such a mentioned case. Also, there could be other base cases that we have not explicitly mentioned here and you should look into them at your own discretion.

Question 1. Please write a function `jump_locate` as discussed above. [6 marks]

Note: Your function should recursively call itself, instead of some inner functions. In the template, we provide the code to track the number of times `jump_locate` *is called. We will use this count to check whether the number of recursive calls by your codes is valid or not. It doesn't mean that your program should have exactly the same function calls as ours, but the number of function calls should be in some reasonable range.*

Sample Executions:

```
>>> a = list(range(1, 10000, 2)) # a is [1, 3, 5, ..., 9997, 9999]
>>> jump_locate(a, 0, len(a)-1, 1, 1001)
500
>>> count
```

```
34
>>> count = 0
>>> jump_locate(a, 0, len(a)-1, 1, 1000)
'Not Found'
>>> count
35
```

Topic 2: Constructing a Binary Tree [7 marks]

In Lecture 8, we discuss about tree. A special kind of tree we want to study in this question is called a *binary tree*. A binary tree is a tree where each node has either a left child, a right child, both a left and a right child, or no children. The node with no children is called a *leaf*, and the node with no parent is called the *root* of the binary tree. We are going to convert from one binary tree representation (in tuple) to another one (also in tuple). To jump ahead, our input tuple is, for example,

```
iTuple = (
  (8,-1,6),
  (7,4,9),
  (5,1,2),
  (1,7,-1),
  (2,3,8),
  (3,-1,0))
```

`iTuple` has 10 nodes here, and the nodes are numbered 0 to 9. Each tuple `(a,b,c)` within `iTuple` is such that `a` is a node with `b` as the left child, and `c` as the right child. If `b` or `c` is `-1`, this means the node has no left or right child, respectively. Notice that a leaf node `a` (with no children) may not be represented explicitly as a tuple `(a,-1,-1)` in `iTuple`.

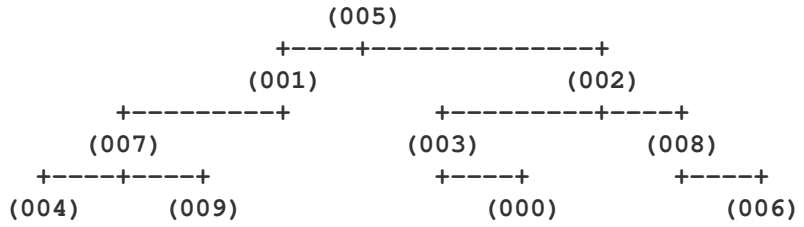
The output tuple (from Question 3 below) that represents the same binary tree from the above example should be as follows:

```
(5, (1, (7, (4, (), ()),
            (9, (), ())),
        ()),
    (2, (3, (),
            (0, (), ())),
        (8, (),
            (6, (), ()))))
```

This representation is closer to what we have in Lecture 8. We have formatted it somehow to show the structure explicitly, but your code (Question 3 below) only need to output the tuple (without the formatting).

For this example, we see the root of the binary tree is 5. The root has two children 1 and 2. And, node 1 has node 7 as it left child but no right child while node 7 in turn has 4 as its left child and 9 as its right child. Similarly, we can check that node 2 has node 3 as its left child and node 8 as its right child while node 3 in turn has a right child 0 (and no left child), and node 8 in turn has a right child 6 (and no left child). The leaves of this binary tree are 4, 9, 0, and 6, each is represented as `(node,(),())`.

This representation actually has lots of parentheses and may be hard to realize what it is. To help you, we have prepared a function `print_tree` that can take the output tuple from the conversion done to show it nicer (though not the best) for ease of understanding of the output – note that you don't need to understand the codes in `print_tree` to do this question, and you can also ignore the use of this function if you don't find it helpful. For our example, the output from `print_tree` on the converted output is:



Question 2. Please write a function `find_root(iTuple)` for the input `iTuple` to output the node number of the root. (For our example, the root is 5, and thus the output is 5.) You can assume that the nodes are numbered sequentially from 0 till $n-1$ where n is the number of nodes in the binary tree. That is, you don't need to worry about any error checking on the input; each input tuple is a valid binary tree. [2 marks]

Question 3. Use the above function `find_root` to write a function `binary_tree(iTuple)`. Your code must use the `find_root` function (where a correct version is provided in Coursemology that you don't worry how you did for the previous question); otherwise, we reserve the right to award lower credit to your answer. [5 marks]

Sample Executions:

```
>>> iTuple = ( (0, 1, 2), )
>>> binary_tree(iTuple)
(0, (1, (), ()), (2, (), ()))
>>> print_tree(binary_tree(iTuple))
     (000)
  +----+----+
(001)     (002)
>>> iTuple = ( (0, 2, -1), (1, 0, -1) )
>>> binary_tree(iTuple)
(1, (0, (2, (), ()), ()), ())
>>> print_tree(binary_tree(iTuple))
          (001)
       +----+
     (000)
  +----+
(002)
>>> iTuple = ( (2, -1, 0), (0, -1, 1) )
>>> binary_tree(iTuple)
(2, (), (0, (), (1, (), ())))
>>> print_tree(binary_tree(iTuple))
(002)
  +----+
     (000)
       +----+
          (001)
>>> iTuple = ( (0, 3, 2), (1, 4, 0) )
>>> binary_tree(iTuple)
(1, (4, (), ()), (0, (3, (), ()), (2, (), ())))
>>> print_tree(binary_tree(iTuple))
     (001)
  +----+---------+
(004)          (000)
            +----+----+
          (003)     (002)
```

Topic 3: Tribes [7 marks]

This is an object-oriented programming exercise. We are going to model tribes in some civilizations. Initially, we have N tribes each is simply an integer from 1 to N, and each is its own leader. That is, the leader of tribe A is also the integer A.

Then, a tribe A can conquer another tribe B to bring B into the leadership of A. That is, whenever we ask for the leader of B, it is either A if A has not previously been conquered, or the same leader of whoever has conquered A (at the point when we ask for B's leader).

If a tribe is conquered by some tribe, it cannot be conquered by other tribes again. You can assume that the inputs are always valid.

You job is to complete the class of `Tribes()` as follows:

```
class Tribes():
    def __init__(self, N):
        # Use a dictionary to capture the leader for each of the N tribes

    def tribe_leader(self, A):
        # Find the leader of tribe A, which is A if no one has conquered it before,
        # or ... (this is related to what you do for the next function, conquer)

    def conquer(self, A, B):
        # Purpose: Tribe A conquers tribe B

    def is_same_tribe(self, A, B):
        # Return True if tribe A and tribe B have the same leader,
        # otherwise return False
```

Sample Executions:

```
>>> N = 100
>>> T = Tribes(N)
>>> T
<__main__.Tribes object at 0x000002123C750898>
>>> T.conquer(10, 20)
>>> T.tribe_leader(20)
10
>>> T.is_same_tribe(20, 11)
False
>>> T.conquer(5, 10)      # 10 now has the same leader as 5
>>> T.conquer(10, 11)     # 10 can still conquer other tribe
>>> T.is_same_tribe(20, 11)
True
```

Question 4. Complete the `Tribes` class. [7 marks]

Topic 4: Snail Traveler [7 marks]

There is a snail traveler living on a cyber-tree. A cyber-tree is a binary tree, where each node has at most 2 children, see Figure 1(a) as an example. A cyber-tree has n nodes labeled from 0 to $n-1$, and the cyber-trees are always rooted at node 0. A cyber-tree can be represented as a `tree` list, where `tree[i]` indicates the parent of node i (`tree[0]` is `-1` since node 0 is the root of the tree and has no parent). For example, the cyber-tree in Figure 1 can be represented as:

```
tree = [-1, 0, 0, 1, 2, 2, 5]
```

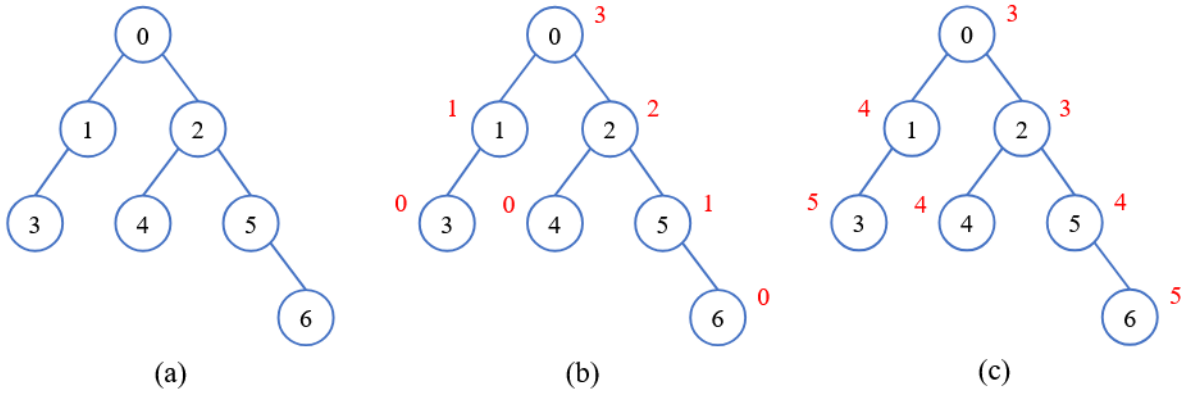


Figure 1: (a) Illustration of a cyber-tree; (b) The distances (shown in red) to the farthest node in the subtree; (c) The distances (shown in red) to the farthest node in the whole tree.

Note that this topic is on a binary tree too just like the one in Topic 2 (Questions 2 and 3). But note that we are using yet a different representation of a binary tree.

The snail traveler loves traveling. It wants to travel all around the world (the cyber-tree)! Since it is impossible for the snail to finish the world travel in one go, the snail comes up with a plan. It plans to first settle down at a node that is close to any other nodes. In other words, it plans to move to a node whose distance to the farthest node is minimized.

On the cyber-tree, the distance between two adjacent nodes is 1, and the distance between two nonadjacent nodes is the length of the path between them. (Note that there is a unique (shortest) path to go between any 2 nodes.) In Figure 1, the distance between node 2 and node 4 is 1, and the distance between node 2 and node 3 is 3. For each node in the cyber-tree, the snail wants to know the distance to the farthest node, so that it can carry out its moving plan.

Let's do something simpler first – What's the distance from each node i to the farthest node in the subtree (rooted at node i)? The subtree rooted at node i consists of node i and all descendants of node i. In Figure 1, the subtree rooted at node 2 consists of the nodes 2, 4, 5 and 6. Among these nodes, the farthest node from node 2 is node 6, with a distance of 2. Figure 1(b) shows the result (in red) for each node in the example tree.

Question 5. Please write a function `subtree_distance` that takes in a `tree` list (represents a cyber-tree) and outputs a list, where the *i*-th element of the output list is the distance from node *i* to the farthest node in the subtree rooted at node *i*. [3 marks]

Sample Execution:

```
>>> subtree_distance([-1, 0, 0, 1, 2, 2, 5])
[3, 1, 2, 0, 0, 1, 0]
```

Now we should be able to compute the distances to the farthest node in the whole tree. In Figure1, node 3 is the farthest node to node 4, and the distance between them is 4. However, for node 3, the farthest node is node 6 and the distance is 5. Figure 1(c) shows all the results (in red) for the example tree.

Question 6. Please write a function `tree_distance` that takes in a `tree` list (represents a cyber-tree) and outputs a list, where the *i*-th element of the output list is the distance from node *i* to the farthest node in the whole tree. [4 marks]

Note: There is a time limitation of 3 seconds for this question. For half of the evaluation test cases, the number of nodes is less than 100, and for the rest of the test cases, the number of nodes is between 100 and 100000. If the time complexity of your solution is not good enough, it is possible that you cannot pass all the test cases.

Hints: For this question, `subtree_distance` *might help as the solution comes partly from these distances. A correct version of* `subtree_distance` *is provided in Coursemology in case you need it (so you don't worry about the correctness of your solution to the previous question).*

Sample Executions:

```
>>> tree_distance([-1, 0, 0, 1, 2, 2, 5])
[3, 4, 3, 5, 4, 4, 5]
>>> tree_distance([-1, 0, 0, 1, 2, 3, 4])
[3, 4, 4, 5, 5, 6, 6]
```

Topic 5: Counting Bugles [5 marks]

In this question, we want to count the Bugles in a $n \times m$ grid. A "Bugle" has the following properties:

- It is an isosceles right triangle, i.e., a right triangle with two equal sides.
- The three vertices all have integer coordinates, i.e., they are grid points.
- It is regular, i.e., at least one of the sides must be horizontal or vertical.

Figure 2 shows all the 28 Bugles over a 3×3 grid.

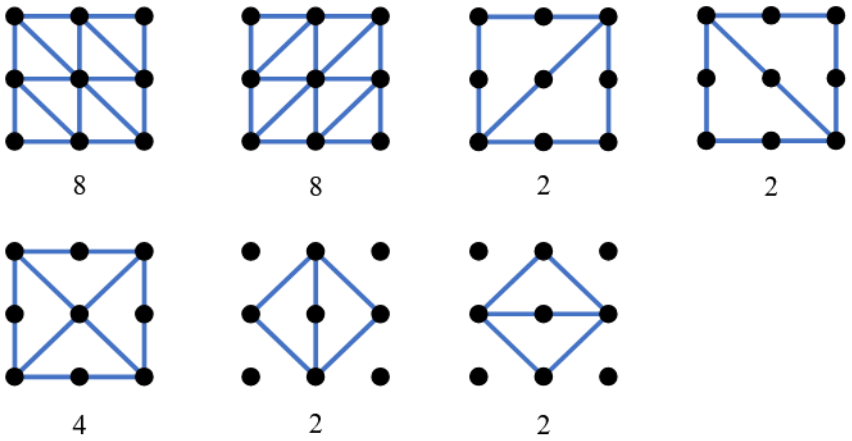


Figure 2: Illustration of the Bugles in a 3×3 grid

Question 7. Please write a Python function `count_bugles` that takes in two integers n and m ($n, m > 1$) and outputs the number of Bugles over a $n \times m$ grid. [5 marks]

Note: There is a time limitation of 3 seconds for this question. For 40% of the evaluation test cases, $n, m \leq 10$, and for the rest of the test cases, $10 \leq n, m \leq 300$. If you use a brute force solution, it is possible that you cannot pass all the test cases.

Hints: For a more intelligent solution, you should find out the various "types" (or shapes or classes) of Bugles, and then sum up the count for each type.

Sample Executions:

```
>>> count_bugles(3, 3)
28
>>> count_bugles(4, 5)
118
```

— E N D O F P A P E R —