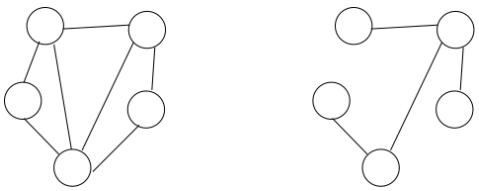
ECE 606, Fall 2021, Assignment 2
Due: Thursday, September 23, 11:59pm

Submission: submit three things: (i) your solutions for the written problems to crowdmark, (ii) the page with your name and student ID for the **[python3]** problem to crowdmark, and, (iii) your **[python3]** file to the appropriate Dropbox on Learn.

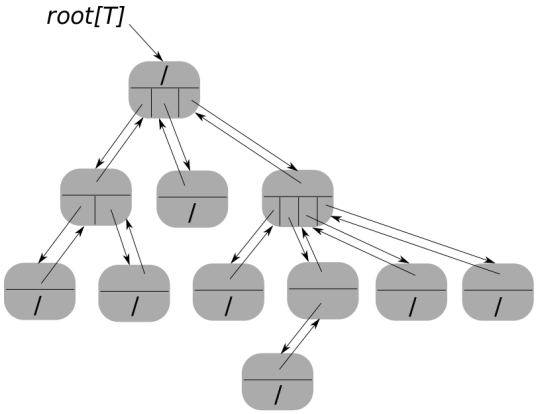
1. This problem refers to the notions of undirected graph, free tree and subgraph, which are defined and discussed in Lecture 2 of your textbook.

   Given a connected undirected graph $G = \langle V, E \rangle$, a *spanning tree* T of G is a particular kind of subgraph of G: $T = \langle V_T, E_T \rangle$ is a free tree with $V_T = V$ and $E_T \subseteq E$. The following picture shows an example of such a G, and a spanning tree of it.



   Prove that if $|E| \geq |V|$, then there exist at least two different spanning trees of G.

2. Consider the following two ways to encode a tree. In Approach 1, if a node has k children, we allocate k pointers in the node, each of which points to a child. An example is shown in the following picture. Note that Approach 1 is similar to the example shown as Figure 10.9, "The representation of a binary tree..." in Lecture 2 of your textbook, except that we have generalized to the possibility of a node having more than two children.



   Approach 2 is the "left-child, right-sibling representation," an example of which is shown as Figure 10.10 in Lecture 2 of your textbook.

   Prove or disprove: to encode a tree T, the total number of pointers we need with Approach 1 is the same as the total number of pointers we need with Approach 2. You should begin your response with 'The statement is true' or 'the statement is false,' and then prove/disprove.

3. This problem pertains to binary search trees. For more context, see the algorithms for binary search trees in Lecture 2 of your textbook, e.g., TREE-SUCCESSOR. The *predecessor* of a node x in a binary search tree of all distinct keys is the node y with the largest key smaller than the key of x. Of course, there can (and does) exist a node in a binary search tree with no predecessor.

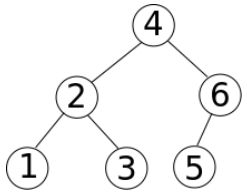
   Prove the following assertion about binary search trees.

   > *In a binary search tree in which the keys are all distinct, if the left subtree of node x is empty and x has a predecessor y in the tree, then y has a right child.*

4. **[python3]** For a set of n keys or values, e.g., integers, we know that there exists a binary search tree of those keys whose height is $\lfloor \log_2 n \rfloor$. In this exercise, you need to design and implement in Python 3 a subroutine, `buildbst`, that builds and returns such a binary search tree. The argument is a Python 3 set of integers.

   The encoding for the binary search tree you should adopt is the following. Your binary search tree should be encoded as a Python 3 list. The first item in the list should be the root of the tree. The left child, if it exists, of the node at index i should be at index $2i + 1$. The right child, if it exists, of the node at index i should be at index $2i + 2$. (Note that indexing in Python 3 starts at 0.)

   As an example, given as argument the set $\{1, 2, 3, 4, 5, 6\}$, you would build a binary search tree of height $\lfloor \log_2 6 \rfloor = 2$. One such tree is the following, which you would encode as the Python 3 list `[4,2,6,1,3,5]`.



   Note that our chosen encoding constrains us to trees that are "full from the left" only. E.g., even though the following would be a valid binary search tree of height 2 for the above set of keys, there is no way to encode it in our encoding, because we are not allowed to have "gaps" in the list.