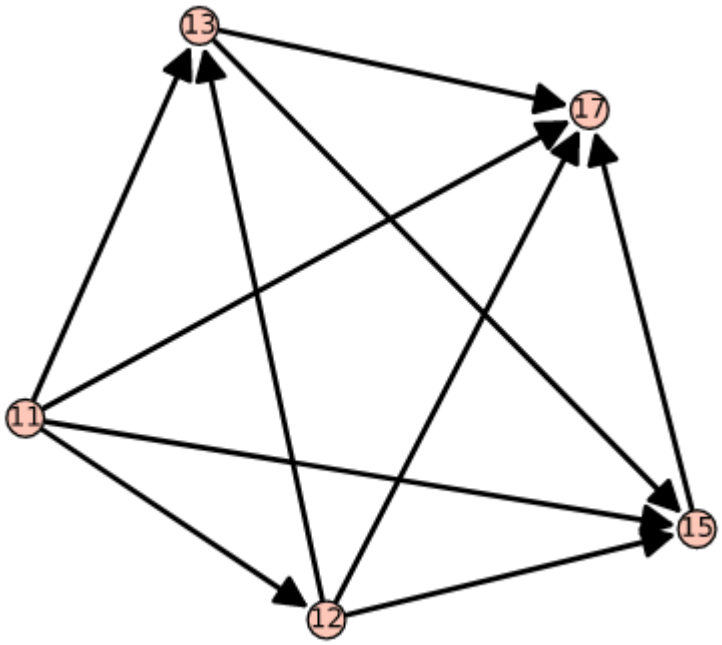
Math 15 - Spring 2017 - Homework 2.1 Solutions

1. (2.1 # 22) Construct a directed network whose vertices represent the numbers 11,12,13,15,17 and whose weights tell how much you must add to get from one vertex to another. Include only edges of positive weight. ANS: Following the instructions at http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/digrap I typed the following commands into sagemath:

```
import igraph                                          # optional - python_igraph
g = igraph.Graph([(0,1),(0,2),(0,3),(0,4),(1,2),\
(1,3),(1,4),(2,3),(2,4),(3,4)], directed=True, \
        vertex_attrs={'name':['11','12','13','15','17']},\
        edge_attrs={'name':['1','2','4','6','1','3','5',\
        '2','4','2'],'weight':[1,2,4,6,1,3,5,2,4,2]})
DG = DiGraph(g)
DG.show()
```

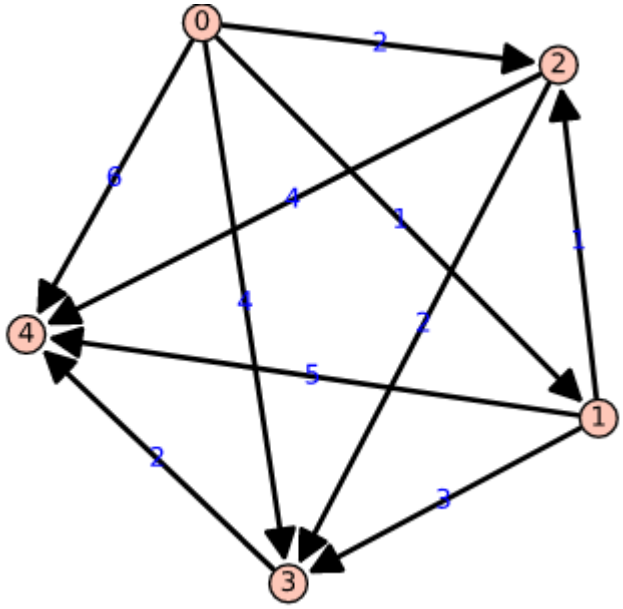
This produced the following graph:



But that didn't label the edge weights, so I came up with these commands (quite different):

```
M = Matrix([[0, 1, 2, 4, 6],[0, 0, 1, 3, 5],[0, 0, 0, 2, 4],\
           [0, 0, 0, 0, 2],[0, 0, 0, 0, 0]]); M
DG5 = DiGraph(M, format='weighted_adjacency_matrix')
H = DG5.plot(edge_labels=True)
H.show()
```

which produced a graph with edge weights labeled, but now I can't label the vertices...



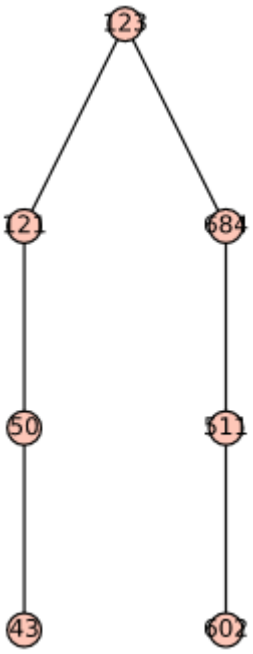
2. (2.1 # 24) Consider the following list of numbers.

$$123, 684, 121, 511, 602, 50, 43$$

(a) Place the numbers, in the order given, into a binary search tree.
The slick way to do this would be to write and implement an algorithm in python for doing just that, and better still, rendering an image of the tree, at least with ascii art. Well, https://pypi.python.org/pypi/pybst is a package to proclaims to accomplish this, though it may not install in Windows? Any road, there's the http://doc.sagemath.org/html/en/reference/combinat/sage/combinat/ordered_tree.html (that's a link) built-in commands that are quite adequate for rendering, once we've decided what it should look like. The root is fixed at 123 by the first element of the list. Then 684 is inserted as the right child and 121 as the left (greater than 123 and less than 123, respectively.) When we're done with this we get a binary search tree we can display in Sagemath in various ways, one of which is which is using the `LabelledOrderTree` function:

```
L43 = LabelledOrderedTree([], label = 43);
L50 = LabelledOrderedTree([L43], label = 50);
L121 = LabelledOrderedTree([L50], label = 121);
R602 = LabelledOrderedTree([], label = 602);
L511 = LabelledOrderedTree([R602], label = 511);
R684 = LabelledOrderedTree([L511], label = 684);
T123 = LabelledOrderedTree([L121,R684], label = 123);
T123.plot()
ascii_art(T123)
```

This produces both a plot:



or the ascii art version:

```
  _123
 /   /
121 684
|   |
50  511
|   |
43  602
```

(b) The height of a binary search tree is the maximum number of edges you have to go through to reach the bottom of the tree, starting at the root. What is the height of the tree in part (a)?
ANS: Height = 3.

(c) Reorder the numbers so that when they are put into a binary search tree, the height of the resulting tree is less than the height of the tree in part (a). Give both your new list and the search tree it produces.
ANS: The following commands

```
RR = LabelledOrderedTree([], label = 684);
RL = LabelledOrderedTree([], label = 511);
LR = LabelledOrderedTree([], label = 121);
LL = LabelledOrderedTree([], label = 43);
R = LabelledOrderedTree([RL,RR], label = 602);
L = LabelledOrderedTree([LL,LR], label = 50);
Rt = LabelledOrderedTree([L,R], label = 121);
#Rt.plot()
ascii_art(Rt)
```

produce this ascii art tree:

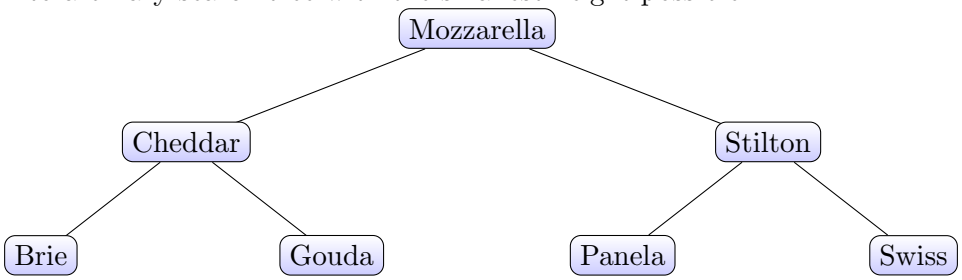
```
    ___121___
   /         /
  50_      _602
 /  /     /   /
43 121   511 684
```

...but I had to comment out the `plot()` operation as it was leading this error: "ValueError: The list associated with vertex 50 has length 3 but d(50)=2"

3. (2.1 # 26) Put the words

   Cheddar Swiss Brie Panela Stilton Mozzarella Gouda

   into a binary search tree with the smallest height possible.



The following Python code:

```
class Node(object):
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
        self.count = 1

    def __str__(self):
        return 'value: {0}, count: {1}'.format(self.value, self.count)

def insert(root, value):
    if not root:
        return Node(value)
    elif root.value == value:
        root.count += 1
    elif value < root.value:
        root.left = insert(root.left, value)
    else:
        root.right = insert(root.right, value)

    return root

def create(seq):
    root = None
    for word in seq:
        root = insert(root, word)

    return root

def search(root, word, depth=1):
    if not root:
        return 0, 0
    elif root.value == word:
        return depth, root.count
    elif word < root.value:
```

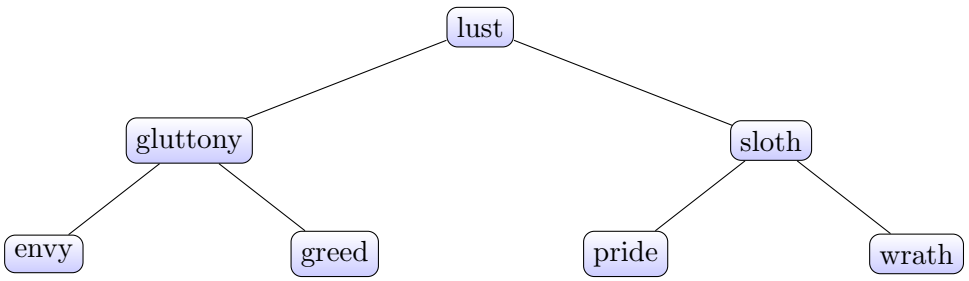
```
            return search(root.left, word, depth + 1)
        else:
            return search(root.right, word, depth + 1)

def print_tree(root):
    if root:
        print_tree(root.left)
        print(root)
        print_tree(root.right)

src = ['Cheddar', 'Swiss', 'Brie', 'Panela', 'Stilton',\
 'Mozzarella', 'Gouda']
tree = create(src)
print_tree(tree)

for word in src:
    print('search {0}, result: {1}'.format(word, search(tree, word)))
```

will produce these results:

```
value: Brie, count: 1
value: Cheddar, count: 1
value: Gouda, count: 1
value: Mozzarella, count: 1
value: Panela, count: 1
value: Stilton, count: 1
value: Swiss, count: 1
search Cheddar, result: (1, 1)
search Swiss, result: (2, 1)
search Brie, result: (2, 1)
search Panela, result: (3, 1)
search Stilton, result: (4, 1)
search Mozzarella, result: (4, 1)
search Gouda, result: (5, 1)
```

4. (2.1 # 28) Place the following words in a binary search tree. Add the words to the tree in the order given.

   lust gluttony greed sloth wrath envy pride



Running the Python code referenced above will now produce these results:

```
value: envy, count: 1
```

```
value: gluttony, count: 1
value: greed, count: 1
value: lust, count: 1
value: pride, count: 1
value: sloth, count: 1
value: wrath, count: 1
search lust, result: (1, 1)
search gluttony, result: (2, 1)
search greed, result: (3, 1)
search sloth, result: (2, 1)
search wrath, result: (3, 1)
search envy, result: (3, 1)
search pride, result: (3, 1)
```