Data Structures Lab #1

Fernando De Santiago

University of Texas at El Paso

Abstract

The following paper will talk about the first lab of Data Structures as well how it is that I decided the approach on how to code and the steps that were considered for its creation. There will be images and figures that will be used in order to emphasis and show what the program is meant to do. While talking about the approaches taken, I will be using vocabulary that should be understood by other computer scientist as well as those informed on computers and programming. There will be progression shots as well that will be used in order to show how the programs used to be and how I modified it to fit the model while also making it seem original and will explain why I named certain variables the way they are.

Data Structures Lab #1

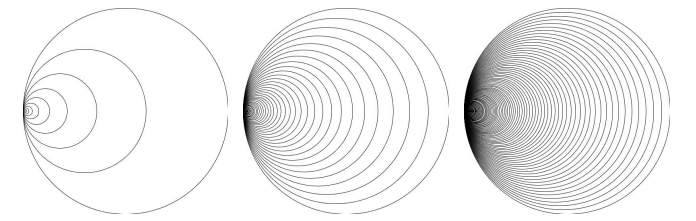
Lab #1 is the focus on creating a recursion method in order to create images using the python matplotlib library. We were given 2 codes that would serve as models and would be a basis for the 4 different methods needed to be able to create the images. The first image consisted of squares along the outer ends of other squares and can be seen on figure one. The second was to create inner circles but that shared a common point and would all collapse into that point (see on figure 2). The thirst program was to create a tree that would divide from its branches and cut its heights by half depending on how many times it’s told to run (figure 3). The final image consisted of creating a cross with circles while inside other circles and can be seen on figure 4.

**Image #1**

**Figure 1
**For the first image(figure 1) I used the model that was provided to us and modified it in order to create the image (figure 5). However, while working with the model I ran into a problem of only the upper right and lower bottom corners appearing. So, I decided to mess with variables and see what was causing it to move right, left, up or down. I then learned that if I wanted to move the images left of right then I would need to change my values on the array. After I learned this fact I then implemented two arrays that were copies of the original and changed the values the by the needed amount in order to have the missing corners appear and meet the requirements.

Figure

## Image #2

 For image two (figure 2) we were given a base mode that consisted of the circles meeting in the middle so I decided to see what would cause it to shift left or right of its variables. I decided that the best method to solve this problem was to leave the model alone and just focus on the x variable that would shift it left by having it add the radius so it would shift right by the approximate amount and would change the radius by a set percentage create the images in figure 2(for final image look at figure 6)

Figure

### Image #3

### 

Figure

For image number 3 we were just shown how it was meant to look like, so I decided to break the program into 2 methods. The two methods that I broke it into were as follows: 1. Drawing the lines right or left 2. How to create the multiple tiers of split offs. The reason for this was in order to keep things organized as well as to keep my code looking clean. For this program I needed to have more variables than before. I used x and y for coordinates at which to start, the width in how far apart the split will be, the height to say how low the lines will be going and tiers for how many levels there will be in the tree. During my time trying to come up with the image I had to look at how far apart I needed my x’s to be and found that both needed to be x plus the width divided by 2 to get the angle needed. And my width had to change at different values in order to get the angle of how much the line had to be away from the origin. One x was width/4 and the other was width\*3/4ths and the tiers would be the y coordinate + height, however during it’s recursion x, y and width had to change in order to make the splits of the trees. Then for my second method it just created the change needed since with each new tear the depth is getting small and smaller due to how many are located on the axis. So, the equation that I came up for that was taking the height and dividing it by tiers and we can see by the result that it worked (figure 7)

**Image #4**

##### .

Figure

For the creation of image 4 it consisted of 2 methods and some changing values in order to make it work. The first method consisted of the properties of the circles in order to find the x and y coordinates as well as the properties needed to get the radius of the circles as it got smaller and how much of the circle was drawn. The second method is what drew the circles as well as shifting them to the places they needed to be. For the second method I needed to calculate how far apart the inner circles were and had to move them over by a variable name newrad I created that would be the radius/3 and that would then be multiplied by 2 to create another variable that would be added or subtracted in order to move it up or down depending if you put it on the x or y axis and would then have the radius of the variable newrad in each new circle which would then make them smaller and smaller by doing it recursively there for looking like figure 8

**Experimental Results**

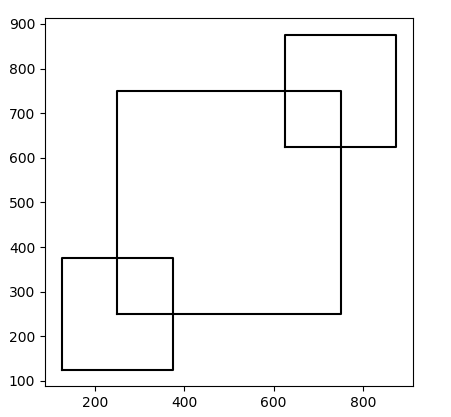
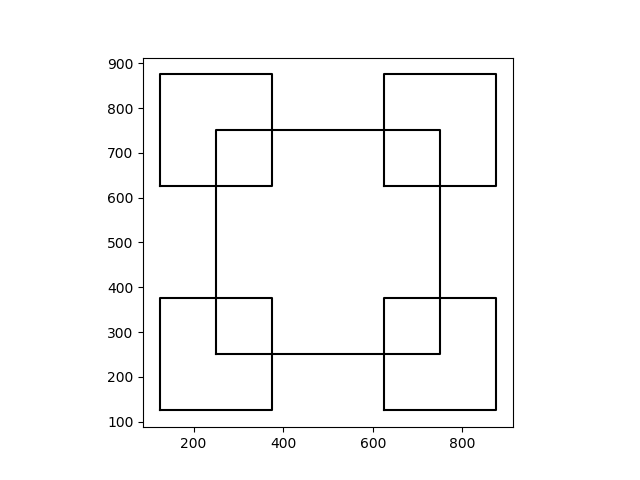
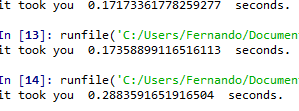
 During my testing I ran into the problem of not knowing how to shift the inner most squares to the corners which resulted with me having the following image. In order to move the squares to the corners I figures that I needed to move array[0] in order to move it left or right and array[1] to move it up and down. Once I got the squares in the right place I tested how long it would take .17 seconds to make the first image figure 9 and the second required square it took .17 seconds and the final required figure took .29 seconds see figure 10.

Figure 10

Figure 5

Figure 9 completed version and meets requirements of figure 1

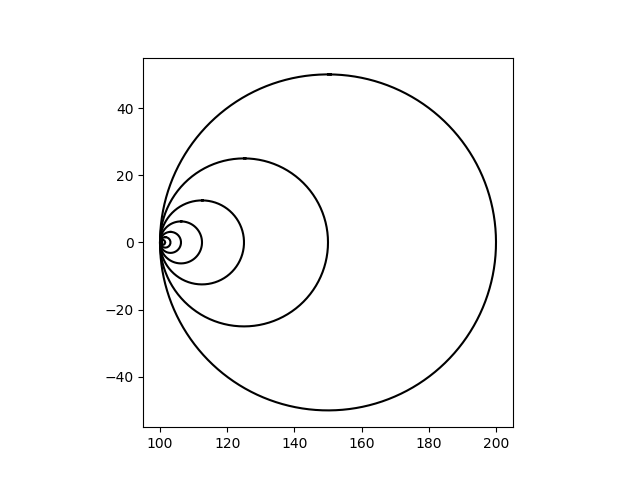
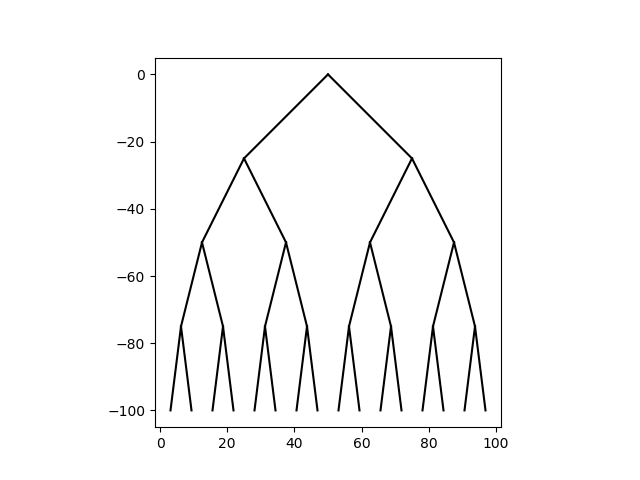
For the second image I used the model that was given to use and only edited one line in order to move the radius to the left and in order to meet the requirements of the three images all that was needed to do was change the number of circles and change the percentage of the used radius. Figure 6 is an example for the first requirement and this only required the x to be x+radius to move left. The run time for this one is as follows circles a. is .35 seconds while circle b. is .16 seconds and the final circle is .25 seconds.

Figure 7

Figure 6

For image 3 I ran into the most problems due to not having first drawn the requirements and instead gone straight to coding which wasn’t the best approach. After this I decided to draw the figure and figure out the dimensions as well as writing the code needed to just draw lines on it’s own. I first started with a straight line and then decided to cut it and break it into two lines after that I tried to see how much I had to alter the x and y in order to create the lines and at the correct angles. This proved most challenging and required lots of tinkering till the right amounts were found and the final image was met(figure 7). The run time for image 3 a. is .14 seconds. Image 3b. is .16 seconds and for 3c the runtime is .23 seconds.

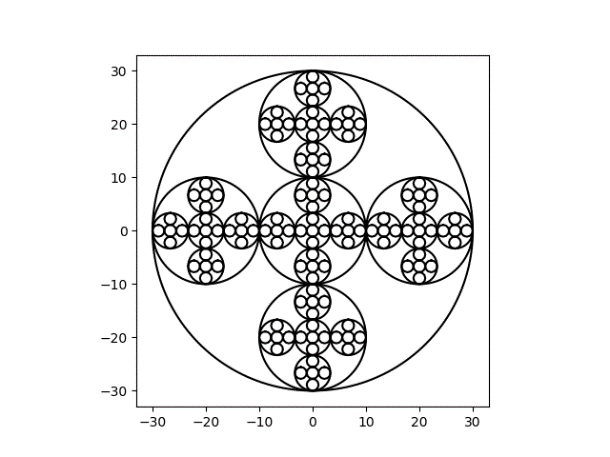
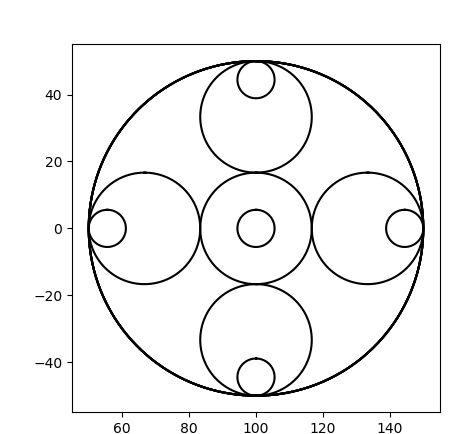
For the final image I decided to start of with a basic circle and then thought of the idea of creating multiple draw circle methods in order to draw the circles and would look like figure 11. I then decided that it seemed like it was to much work to run and instead decided to trash that whole method and decided to create a equations to see how far apart the circles are and how they should be moved. This resulted in me creating variables for new radii and another variable to make them smaller in order to fit them within the circles. However, while coding that there were some issues that I had which would convert the inner most circles to hexagons but it turned out that I just needed to change the radius in the original draw circles and I ended up making a coding error twice which crashed my system due to it trying to process something that is impossible. The final image had the following run times image 4a. .22 seconds 4b. 45 seconds and 4c. is 1.74 seconds.

Figure 11

Figure 8

**Conclusion**

During this lab I learned that my basic approach of just coding and meeting requirements would not always work, and I needed to come up with a plan. I also learned that for some segments it was easier to break it into different methods in order to keep code clean and organized. There was also the timing of the program in which I learned that by overthinking and adding more lines than needed it would then be slower and delayed so I cut any unnecessary lines or extra. While this lab was the first I found it very challenging due to not having learned the library so I learned that in order to do something I need to be familiar with the program and it’s multiple tools that can be used in order to make my job easier as well as making it run more efficiently.

**Appendix**

"""

Created on Mon Feb 4 08:17:26 2019

@author: Fernando

"""

import numpy as np

import matplotlib.pyplot as plt

import math

def draw\_squares(ax,n,perimeter,percent):

if n>0:

area = perimeter\*percent

perimeter2=np.copy(perimeter)#creating a copy of the array perimeters

perimeter2[:,1]+=1000#adding 1000 to the y column to move it up

perimeter3=np.copy(perimeter)#creating another copy of the array perimeters

perimeter3[:,0]+=1000#adding 1000 to move it to the right

area2= perimeter2\*percent#getting the area for the new squares in the corners

area3=perimeter3\*percent#getting area for the new squares in the corners

ax.plot(perimeter[:,0],perimeter[:,1],color='k')

draw\_squares(ax,n-1,area+500,percent)#upper right corner

draw\_squares(ax,n-1,area2,percent)#upper left corner

draw\_squares(ax,n-1,area,percent)#bottom left corner

draw\_squares(ax,n-1,area3,percent)#bottom right corner

plt.close("all")

orig\_size = 1000

perimeter = np.array([[250,250],[250,750],[750,750],[750,250],[250,250]])

fig, ax = plt.subplots()

draw\_squares(ax,2,perimeter,.50)

ax.set\_aspect(1.0)

plt.show()

fig.savefig('squares.png')

fig,ax1=plt.subplots()

draw\_squares(ax1,3,perimeter,.5)

ax1.set\_aspect(1.0)

plt.show()

fig.savefig('squares1.png')

fig,ax2=plt.subplots()

draw\_squares(ax2,4,perimeter,.5)

ax2.set\_aspect(1.0)

plt.show()

fig.savefig('squares2.png')

#Square ends here

#circle Starts here

def circle(center,rad):

n = int(4\*rad\*math.pi)#radius of each circle

t = np.linspace(0,6.3,n)#creating the circles

x = center[0]+rad\*np.sin(t)

y = center[1]+rad\*np.cos(t)

return x,y

def draw\_circles(ax,n,center,radius,w):

if n>0:

x,y = circle(center,radius)

ax.plot((x+radius),y,color=(0,0,0))#shifts origin to the left

draw\_circles(ax,n-1,center,radius\*w,w)

fig, ax3 = plt.subplots()

draw\_circles(ax3, 10, [100,0], 50,.5)

ax3.set\_aspect(1.0)

plt.show()

fig.savefig('circles.png')

fig, ax4 = plt.subplots()

draw\_circles(ax4, 20, [100,0], 50,.7)

ax4.set\_aspect(1.0)

plt.show()

fig.savefig('circles2.png')

fig, ax5 = plt.subplots()

draw\_circles(ax5, 50, [100,0], 50,.9)

ax5.set\_aspect(1.0)

plt.show()

fig.savefig('circles3.png')

#circle ends here

#binary tree starts here

def draw\_line(ax, x1,y1,x2,y2):

n = int(max( abs(x1-x2), abs(y1-y2)) )#the range of the lines

x = np.linspace(x1,x2,n)

y = np.linspace(y1,y2,n)

ax.plot(x,y,color='k')

def recur\_tree(ax, x, y, width, height, tiers):

if tiers > 0:

draw\_line(ax, x+width/2,y, x+width/4,y+height)#drawing line to the left down

draw\_line(ax, x+width/2,y, x+width\*3/4,y+height)#drawing line to the right down

tiers -= 1

recur\_tree(ax, x, y+height, width/2, height, tiers) #moving the tree down slowly

recur\_tree(ax, x+width/2, y+height, width/2, height, tiers)#mocing the tree in different directions slowly

def tree(ax, x, y, width, height, tiers):

tier\_height = height/tiers #how much each tier goes down

recur\_tree(ax, x, y, width, tier\_height, tiers)

fig, ax6=plt.subplots()

tree(plt,0,0,100,-100,3) #plot starting at [0,0] and going to [100,-100] and doing it 3 times each time getting smaller in tier but lower down

ax6.set\_aspect(1.0)

plt.show()

fig.savefig('tree.png')

fig, ax7=plt.subplots()

tree(plt,0,0,100,-100,4)#plot starting at [0,0] and going to [100,-100] and doing it 4 times each time getting smaller in tier but lower down

ax7.set\_aspect(1.0)

plt.show()

fig.savefig('tree2.png')

fig, ax8=plt.subplots()

tree(plt,0,0,100,-100,5)#plot starting at [0,0] and going to [100,-100] and doing it 5 times each time getting smaller in tier but lower down

ax8.set\_aspect(1.0)

plt.show()

fig.savefig('tree3.png')

#tree ends here

#cirlces number 2 starts here

def circle2(center,rad):

n = int(4\*rad\*math.pi)#the radius of each circle

t = np.linspace(0,6.3,n)#creating the circle

x = center[0]+rad\*np.sin(t)#x radius

y = center[1]+rad\*np.cos(t)#y radius

return x,y

def draw\_circles2(ax,n,center,radius):

if n>0:

x,y = circle2([center[0],center[1]],radius)# the origin points

ax.plot(x,y,color='k')

newrad=radius/3#radius needed to make the circles smaller

offset=newrad\*2#radius used to move the circles up,down,left or right

draw\_circles2(ax,n-1,[center[0],center[1]],newrad)#center circle

draw\_circles2(ax,n-1,[center[0]-offset,center[1]],newrad)#moving the circle right

draw\_circles2(ax,n-1,[center[0]+offset,center[1]],newrad)#movinf the cirlce left

draw\_circles2(ax,n-1,[center[0],center[1]-offset],newrad)#moving the circle up

draw\_circles2(ax,n-1,[center[0],center[1]+offset],newrad)#moving circle down

fig, ax9 = plt.subplots()

draw\_circles2(ax9, 3, [0,0], 30)

ax9.set\_aspect(1.0)

plt.show()

fig.savefig('circles4.png')

fig, ax10 = plt.subplots()

draw\_circles2(ax10, 4, [0,0], 30)

ax10.set\_aspect(1.0)

plt.show()

fig.savefig('circles5.png')

fig, ax11 = plt.subplots()

draw\_circles2(ax11, 5, [0,0], 30)

ax11.set\_aspect(1.0)

plt.show()

fig.savefig('circles6.png')