CS 2302 Data Structures

Spring 2019

MW 10:30-11:50 in CCSB 1.0202

LAB # 1

Claudio Angel Garcia

Instructor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

IA: Eduardo Lara

Peer leader: Erick Macik

Introduction

The problem we are trying to solve is, how do you do a recursive method in Python that can draw squares, and then draw another square using the four vertices as the new origin to keep drawing more squares depending on the amount of repetitions we want and so on?

Another problem is, how do you implement a recursive method that can draw a circle and then draws another circle but this time smaller and moved closer to the right of the previous circle.

The third problem is, how do you create a recursive program that can draw a binary-like tree depending on how many repetitions you want.

The fourth problem is, how do you implement a recursive method that can draw five circles that are in the left, right, up and down part inside a circle?

We have these four problems and we want to solve them specifically using recursive calls which are basically a form of loop using methods. Inside this method we call the same method but with changes on the base case and other variables so it can stop the loop at some point.

Solution

I approach the solutions mainly by tracing them in a white board. For the squares I had to think using basic geometry. By looking at the given figures that we were supposed to do, I draw them above a Cartesian plane so I can view where to place the squares. After that, I started to put a coordinate to every vertex of the square thus giving me the places where I can draw the next square, for example, having a square in which the bottom left vertex its at [0, 0] I can now take that as my next origin for my next square although it wasn’t that easy. So what I did is change every vertex using the 75% and 25% of the big square or previous square, so for the bottom left square the vertex point [0, 0] (which is the x axis) I had to add that the 75%, and for the vertex point[0, 1] (which is the y axis) I had to subtract that the 25% and so on with the same method for all the squares, the top left, top right, and bottom right.

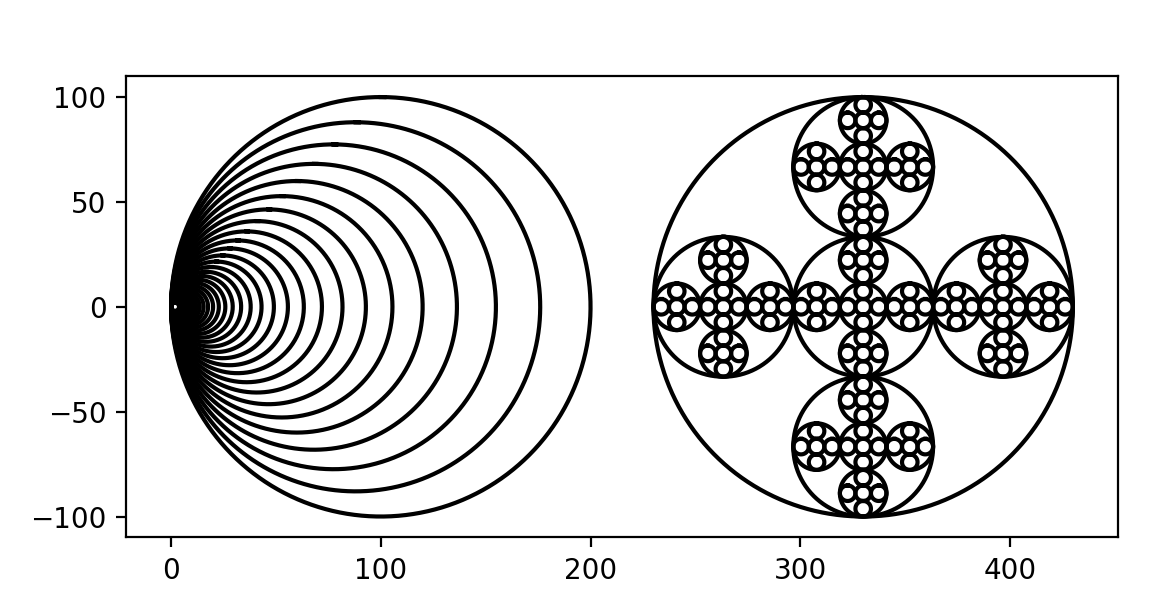
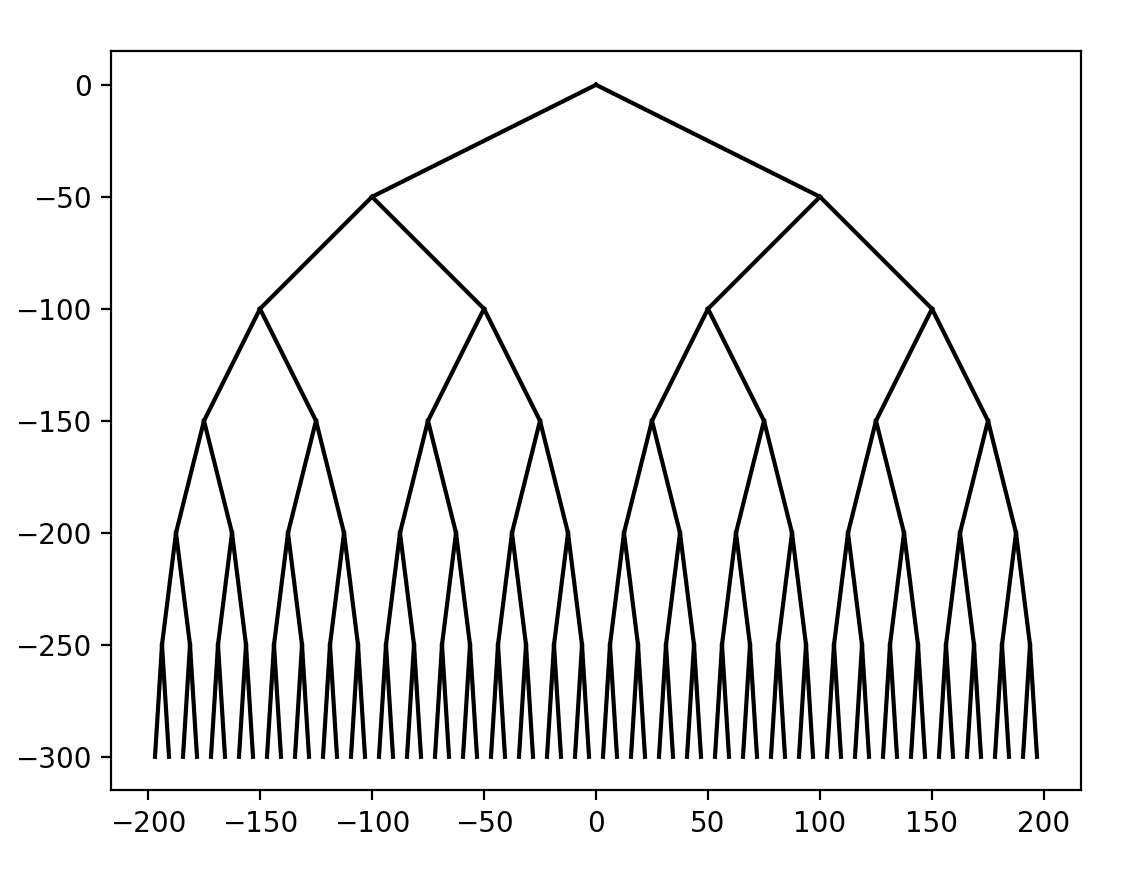
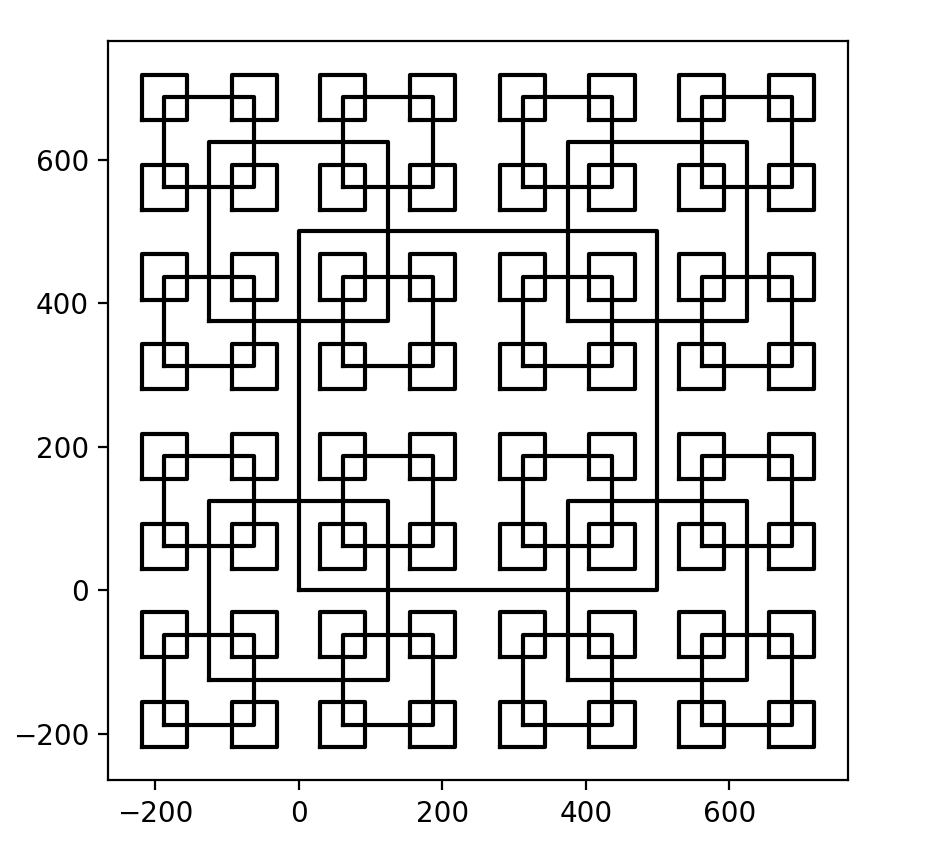
For the first cycles problem we just had to move the next circle a little bit too the right and make it smaller, so what I did was to multiply radius times the width and make that my new x value for the center on the recursive call. The given code already did the circle smaller and smaller on each recursive call by multiplying the radius times the width. For the fourth problem, which is the circle that has five circles inside, I struggle a lot because I tried to make each individual circle a method, so first I did the left and then the right and so on, but the main problem is that I didn’t had a constant that I can use to change the center, the only one that was working was the left circle. Thus, I found that I can make a different circle each time I do a recursive call, so what I did was to make five different recursive calls, each one with different center values, for the right and left ones I only change the x coordinate of the center by adding the radius time 1.5 which is exactly the half radius of the big circle or the previous circle, that for the right one, and the same for the left circle but this one I subtracted the same equation. For upper and down circles did exactly the same but on the y axis of the center of the circle.

For the trees I used the same approach of the squares at first, treating everything with percentages and vertices, but trying so many things I ended up discarding that method and trying the same approach of the circles. First I call the method sending a value that will change the x coordinate and another for the y coordinate that will have to be x/2, I also send a x and y value that are equal to zero, this to make changes on the recursive calls. To make a line you only need two vertices that connect, so I did two different arrays, one to hold two vertices to make right lines and one to make the left lines. The values for the arrays are [x, y] (0 and 0) because we want them to be in the same position and from that draw a line to a different point. For the values on the second variable of the array I put [x-x\_change, y-y\_change] (0-100 and 0-50) and then plot those two values and thus, making my right line, same approach is applied for the left line. After that I did the recursive call sending my x\_change over two to get the half and make the next line smaller in width, and my y change will stay the same. And for the new values for x and y I subtract x and x\_change and y and y\_change and these one would be my new x and y coordinates so it keeps on moving downwards and to the left. Same on the second recursive call but the x adding to x\_change.

Experimentation

Squares with input: 3, Trees with input: 6 and circles 1 and 2 with inputs: 4

Running times with these input values:

* Squares: 2.1894140243530273 seconds
* Trees: 1.8818509578704834 seconds
* Circle 1 and 2 together: 2.0845980644226074 seconds

Conclusion

With this lab I learned very strongly how to draw at least from vertex to vertex with the matplotlib.pyplot library using numpy arrays. I also learned a better way to implement recursive methods. I learned about a process of tracing everything before you translating to a code to get a much better result with fewer errors.

Appendix

CIRCLES 1 AND 2

"""

Created on Thu Feb 7 10:56:22 2019

@author: claudiogarcia

CS 2302 Data Structures Spring 2019

MW 10:30-11:50 in CCSB 1.0202

Instructor: Olac Fuentes

TAs: Anindita Nath, Maliheh Zargaran. IA: Eduardo Lara

Peer Leader: Erick Macik

Lab 1

Claudio Garcia 80628314

The purpose of this program is to draw a circles that keeps on moving to the left,

and the second method draws circles to left, right, up, and right depending on users

number input which will translate to number of repetitions.

"""

import matplotlib.pyplot as plt

import numpy as np

import math

import time

def circle(center,rad):

n = int(4\*rad\*math.pi)

t = np.linspace(0,6.3,n)

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,y,color='k')

draw\_circles(ax,n-1,[radius\*w, 0],radius\*w,w) # center x = 90...

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

if n>0:

x,y = circle(center,radius)

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

draw\_circles2(ax,n-1,center,radius/3,w) #Center circle

draw\_circles2(ax,n-1,[center[0]+radius/1.5, center[1]],radius/3,w) # x to the right

draw\_circles2(ax,n-1,[center[0]-radius/1.5,center[1]],radius/3,w) # x to the left

draw\_circles2(ax,n-1,[center[0],center[1]+radius/1.5],radius/3,w) # y to up

draw\_circles2(ax,n-1,[center[0],center[1]-radius/1.5],radius/3,w) # y to down

start\_time = time.time()

rep = int(input('How many repetitions you want?: '))

plt.close("all")

fig, ax = plt.subplots()

c = [100, 0]

c2 = [330, 0]

draw\_circles(ax, rep\*8, c, 100,.88)

draw\_circles2(ax, rep, c2, 100,.9)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('circles.png')

print("--- %s seconds ---" % (time.time() - start\_time))

SQUARES

"""

Created on Thu Feb 7 13:26:58 2019

@author: claudiogarcia

CS 2302 Data Structures Spring 2019

MW 10:30-11:50 in CCSB 1.0202

Instructor: Olac Fuentes

TAs: Anindita Nath, Maliheh Zargaran. IA: Eduardo Lara

Peer Leader: Erick Macik

Lab 1

Claudio Garcia 80628314

The purpose of this program is to draw a square and then grab that square vertices

and use them as origin to keep drawing squares depending on users number input

which will translate to number of repetitions.

"""

import numpy as np

import matplotlib.pyplot as plt

import time

def draw\_squares(ax,n,ver,w):

########## bottom left ###########

if n>0:

axChange = [w\*.25,w\*.75] #what portion the x and y coordinates are changing

bot\_right = np.copy(ver)

bot\_right[0] = bot\_right[0] - axChange[0]

bot\_right[1][0] = bot\_right[1][0] - axChange[0]

bot\_right[1][1] = bot\_right[1][1] - axChange[1]

bot\_right[2] = bot\_right[2] - axChange[1]

bot\_right[3][0] = bot\_right[3][0] - axChange[1]

bot\_right[3][1] = bot\_right[3][1] - axChange[0]

bot\_right[4] = bot\_right[4] - axChange[0]

ax.plot(bot\_right[:,0],bot\_right[:,1],color='k')

draw\_squares(ax,n-1,bot\_right,w/2)

########## bottom right ###########

bot\_left = np.copy(ver)

bot\_left[0][0] = bot\_left[0][0] + axChange[1]

bot\_left[0][1] = bot\_left[0][1] - axChange[0]

bot\_left[1][0] = bot\_left[1][0] + axChange[1]

bot\_left[1][1] = bot\_left[1][1] - axChange[1]

bot\_left[2][0] = bot\_left[2][0] + axChange[0]

bot\_left[2][1] = bot\_left[2][1] - axChange[1]

bot\_left[3][0] = bot\_left[3][0] + axChange[0]

bot\_left[3][1] = bot\_left[3][1] - axChange[0]

bot\_left[4][0] = bot\_left[4][0] + axChange[1]

bot\_left[4][1] = bot\_left[4][1] - axChange[0]

ax.plot(bot\_left[:,0],bot\_left[:,1],color='k')

draw\_squares(ax,n-1,bot\_left,w/2)

########## upper right ###########

up\_right = np.copy(ver)

up\_right[0][0] = up\_right[0][0] - axChange[0]

up\_right[0][1] = up\_right[0][1] + axChange[1]

up\_right[1][0] = up\_right[1][0] - axChange[0]

up\_right[1][1] = up\_right[1][1] + axChange[0]

up\_right[2][0] = up\_right[2][0] - axChange[1]

up\_right[2][1] = up\_right[2][1] + axChange[0]

up\_right[3][0] = up\_right[3][0] - axChange[1]

up\_right[3][1] = up\_right[3][1] + axChange[1]

up\_right[4][0] = up\_right[4][0] - axChange[0]

up\_right[4][1] = up\_right[4][1] + axChange[1]

ax.plot(up\_right[:,0],up\_right[:,1],color='k')

draw\_squares(ax,n-1,up\_right,w/2)

########## upper left ###########

up\_left = np.copy(ver)

up\_left[0] = up\_left[0] + axChange[1]

up\_left[1][0] = up\_left[1][0] + axChange[1]

up\_left[1][1] = up\_left[1][1] + axChange[0]

up\_left[2] = up\_left[2] + axChange[0]

up\_left[3][0] = up\_left[3][0] + axChange[0]

up\_left[3][1] = up\_left[3][1] + axChange[1]

up\_left[4] = up\_left[4] + axChange[1]

ax.plot(up\_left[:,0],up\_left[:,1],color='k')

draw\_squares(ax,n-1,up\_left,w/2)

start\_time = time.time()

plt.close("all")

orig\_size = 500

ver = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]])

fig, ax = plt.subplots()

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

rep = int(input('How many repetitions you want?: '))

draw\_squares(ax,rep,ver,orig\_size) #we take the original size since in this case it means the length of the square

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('squares.png')

print("--- %s seconds ---" % (time.time() - start\_time))

TREES

"""

Created on Sun Feb 10 13:54:02 2019

@author: claudiogarcia

CS 2302 Data Structures Spring 2019

MW 10:30-11:50 in CCSB 1.0202

Instructor: Olac Fuentes

TAs: Anindita Nath, Maliheh Zargaran. IA: Eduardo Lara

Peer Leader: Erick Macik

Lab 1

Claudio Garcia 80628314

The purpose of this program is to draw a binary-like tree depending on users

number input which will translate to number of repetitions.

"""

import numpy as np

import matplotlib.pyplot as plt

import time

def draw\_trees(ax, n, x\_change, y\_change, x, y ):

if n>0:

#changing the center x to right and left. y changes always downwards

right = np.array([[x, y], [x - x\_change, y - y\_change]])

left = np.array([[x, y], [x + x\_change, y - y\_change]])

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

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

draw\_trees(ax, n-1, x\_change / 2, y\_change, x - x\_change, y - y\_change)

draw\_trees(ax, n-1, x\_change / 2, y\_change, x + x\_change, y - y\_change)

start\_time = time.time()

rep = int(input('How many repetitions you want?: '))

plt.close("all")

fig, ax = plt.subplots()

ax.axis('on')

draw\_trees(ax, rep, 100, 50, 0, 0)

print("--- %s seconds ---" % (time.time() - start\_time))

ax.set\_aspect(1.0)

plt.show()

fig.savefig('lines.png')