```
# -*- coding: utf-8 -*-
#!/usr/bin/python
# Juracy Filho

from Image import new as Image
from ImageDraw import ImageDraw
from ImageColor import colormap


class Canvas(ImageDraw):
    """Especialização da classe ImageDraw com suporte a elipses com texto"""

    def text(self, pos, texto):
        """Sobreescreve o método text, porém centralizando"""
        tam = self.textsize(texto)
        newpos = (pos[0] - tam[0] / 2, pos[1] - tam[1] / 2)
        ImageDraw.text(self, newpos, texto)

    def ellipseText(self, coords, color, texto):
        # esq, topo, dir, bottom
        # esq, top
        centro = (coords[0] + (coords[2] - coords[0]) / 2,
                  coords[1] + (coords[3] - coords[1]) / 2)
        self.ellipse(coords, color)
        self.text(centro, texto)


class Coord(tuple):
    """Classe para coordenadas com um suporte aritmético mínimo """

    def __add__(self, valor):
        return Coord((self[0] + valor[0],
                      self[1] + valor[1],
                      self[2] + valor[0],
                      self[3] + valor[1]))

    def __sub__(self, valor):
        valor = (-1 * valor[0], -1 * valor[1])
        return self.__add__(valor)

    def center(self):
        return ((self[2] - self[0]) / 2 + self[0],
                (self[3] - self[1]) / 2 + self[1])


class DrawArvore(object):
    """Classe responsável por desenhar uma árvore binária"""
    largura = 40
    altura = 40
    topo = 0
    gap_horizontal = 15
    gap_vertical = 30

    def __init__(self, profundidade):
        self.min = None
        self.max = None
        self.profundidade = profundidade
        self.largura_total = ((2**(profundidade-1)+6) *
                              (self.largura+self.gap_horizontal))
        self.altura_total = profundidade * self.altura + (profundidade - 1) * self.gap_vertical

        self.im = Image("RGB", (self.largura_total, self.altura_total))
        self.draw = Canvas(self.im)

    def execute(self, raiz):
        coord_central = self.largura_total / 2
        coord_inicial = Coord((coord_central - (self.largura / 2),
                               self.topo,
                               coord_central + (self.largura / 2),
                               self.topo + self.altura))
        self.drawNo(raiz, coord_inicial)
        return self.im.crop((self.min, 0, self.max, self.altura_total))
```

```
    def drawNo(self, no, coord_inicial, nivel = 1):
        if not self.min or self.min > coord_inicial[0]:
            self.min = coord_inicial[0]

        if not self.max or self.max < coord_inicial[2]:
            self.max = coord_inicial[2]

        left_side = coord_inicial - ((self.gap_horizontal * (2**(self.profundidade-nivel-1)) - 1) +
self.largura, (self.gap_vertical + self.altura) * -1)
        right_side = coord_inicial + ((self.gap_horizontal * (2**(self.profundidade-nivel-1)) - 1) +
self.largura, self.gap_vertical + self.altura)
        first_line = list(coord_inicial.center())
        first_line += left_side.center()
        second_line = list(coord_inicial.center())
        second_line += right_side.center()
        if no.menor:
            self.draw.line(first_line, colormap['green'])
            self.drawNo(no.menor, left_side, nivel + 1)

        if no.maior:
            self.draw.line(second_line, colormap['green'])
            self.drawNo(no.maior, right_side, nivel + 1)

        self.draw.ellipseText(coord_inicial, colormap['green'], str(no.valor))

if __name__ == '__main__':
    from arvore_binaria import Arvore
    from random import randint
    import sys
    try:
        elementos = int(sys.argv[1])
    except:
        # Quantidade default de nós a gerar
        elementos = 10

    arv = Arvore()
    for i in range(elementos):
        r = randint(1, 500)
        if not arv.procurar(r):
            arv.adicionar(r)

    draw_arvore = DrawArvore(arv.profundidade)
    print "profundidade: ", arv.profundidade
    print draw_arvore.altura_total, draw_arvore.largura_total

    im = draw_arvore.execute(arv.raiz)
    im.show() #save("arvore.png")
```