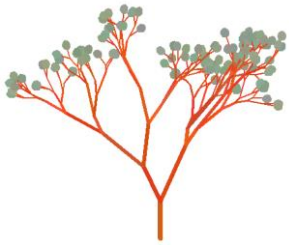
Árboles fractales con Turtle en Python

Mauricio Avilés Cisneros
Instituto Tecnológico de Costa Rica, Escuela de Ingeniería en Computación
maviles@itcr.ac.cr

Resumen

Se presentan funciones que utilizan recursión para la construcción de un árbol fractal binario utilizando el lenguaje Python 3 y la biblioteca de gráficos Turtle. Se presentan modificaciones a las funciones para agregar aleatoriedad a la generación de fractales para obtener dibujos de árboles que se asemejan más a los árboles encontrados en la naturaleza. Finalmente se presentan los resultados de pruebas realizadas.



Introducción

Los fractales son objetos geométricos que tienen la particularidad de que su estructura o forma básica se repite dentro del mismo objeto en diferentes escalas. El término fue acuñado por Benoît Mandelbrot en 1975, pero las raíces de la idea de fractal se remontan al siglo XVII con las primeras nociones de recursión.

A pesar de que son un concepto matemático, los fractales también se encuentran en la naturaleza. Se pueden apreciar en la forma de las brócolis, repollos, hojas de helechos, copos de nieve, el cauce de los ríos, en la forma de los árboles, entre otros ejemplos. El concepto clave acá es la autosimilaridad, que se da cuando una forma está compuesta por copias más pequeñas de la misma figura.

En este artículo se hablará sobre la construcción de uno de los fractales más simples que existe, los árboles fractales, por medio de la utilización de gráficas tortuga en el lenguaje Python en su versión 3.2.3.

La idea básica de este fractal es muy simple, se empieza con una línea que es el tronco del árbol y que se divide en dos ramas más pequeñas. Ese mismo proceso se repite para cada una de las ramas hasta llegar a una cantidad límite de divisiones (profundidad del árbol). Como las ramas del árbol siempre se dividen en dos sub-ramas, el resultado es un árbol binario. Y, dado que se da la repetición de un mismo proceso a diferentes escalas, el concepto de recursión es perfecto para la implementación de las rutinas de dibujo.

Las gráficas tortuga es el mecanismo que se utilizará para describir las figuras geométricas. Este concepto tuvo auge a partir de la década de los 60 cuando el Dr. Seymour Papert lo añadió al lenguaje de programación LOGO, que fue utilizado durante las siguientes décadas para enseñar nociones de programación a jóvenes de primaria y secundaria en todo el mundo.

Gráficas tortuga

Las gráficas tortuga se basan en el concepto de "tortuga", que es similar al concepto de punto en la geometría euclidiana, pero posee otras características:

1. Ubicación en el plano cartesiano
2. Una dirección
3. Un lápiz, que a su vez tiene múltiples características

La tortuga responde a comandos que se ejecutan en relación a su posición y estado actual, como por ejemplo "adelante 20 espacios" o "girar a la derecha 90 grados". Al desplazarse, la tortuga dibuja una línea sobre la trayectoria recorrida. De esta forma, por medio de desplazamientos y giros, principalmente, se pueden representar figuras geométricas de todo tipo.

Para hacer este tipo de gráficas en Python no es necesario descargar ninguna biblioteca adicional ya que el mismo lenguaje incluye el paquete "turtle" que permite construir figuras de una manera fácil.

```
import turtle
```

Luego de importar esta biblioteca, cualquier comando que se le dé a la Tortuga abre automáticamente una ventana donde se puede observar la tortuga en movimiento dibujando su trayectoria. El siguiente código es la definición de un simple cuadrado utilizando la geometría de la tortuga.

```
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
```

Implementación

Como se mencionó anteriormente, el árbol fractal que se desea dibujar, al igual que la mayoría de fractales, se puede construir por medio de definiciones recursivas. Para esto se utilizará una función recursiva que va a recibir dos parámetros, el primero será el tamaño del tronco inicial que se va a dibujar y el segundo es la profundidad del árbol.

La profundidad indica cuántas veces hay que repetir el dibujo del árbol antes de terminar. Por ejemplo, un árbol de profundidad tres inicia con un tronco que se divide en dos, cada una de las dos ramas resultantes se divide de nuevo en dos ramas, y a su vez estas cuatro ramas se dividen en dos cada una, generando un total de ocho ramas. Al contar la cantidad de veces que se dividen las ramas, se obtiene la profundidad del árbol.

```
def arbol(tam, prof):
    if prof==0:
        return
    else:
        turtle.forward(tam)
        turtle.left(45)
        arbol(tam*2/3, prof-1)
        turtle.right(90)
        arbol(tam*2/3, prof-1)
        turtle.left(45)
        turtle.back(tam)
```

El caso base de esta función es cuando el parámetro `prof` vale cero, que indica que debe detenerse la recursión y no seguir dibujando más ramas en el árbol. En este caso la función no realiza ninguna tarea y termina su ejecución.

En caso contrario la función dibuja el tronco del árbol, avanzando el tamaño especificado por el parámetro `tam`. Luego se realiza un giro a la izquierda de 45 grados y se hace una llamada recursiva a la función con dos tercios del tamaño original y restándole uno a la profundidad. Esto dibujaría la rama izquierda del árbol.

Luego, asumiendo que la llamada recursiva deje a la tortuga en la misma posición en que estaba antes de dibujar la rama izquierda, se hace el giro opuesto de 90 grados hacia la derecha para dibujar la rama de la derecha. Esto también se realiza por medio de una llamada recursiva.

Finalmente, se regresa la tortuga a su posición inicial haciendo un giro de 45 grados a la izquierda y retrocediendo al punto donde inició el dibujo del árbol.

Es importante recalcar que es indispensable que la tortuga regrese al punto de partida, debe recordarse que la función es recursiva y es la misma función la que se utiliza para dibujar las ramas del árbol. Si no se regresa la tortuga a la posición inicial, entonces cuando se dibujen las ramas del árbol, no se puede esperar que la tortuga esté en el mismo punto luego de dibujar la primera, lo que es necesario para dibujar la segunda.

```
turtle.speed(0)
turtle.left(90)
arbol(120, 10)
turtle.done()
```

Si se invoca esta función con parámetros iniciales de 120 pixeles de tamaño y 10 niveles de profundidad, se obtiene el resultado de la figura 1. Antes de invocar la función, se dan dos instrucciones a la tortuga para que utilice la mayor velocidad de dibujo y se ubique en dirección "norte".

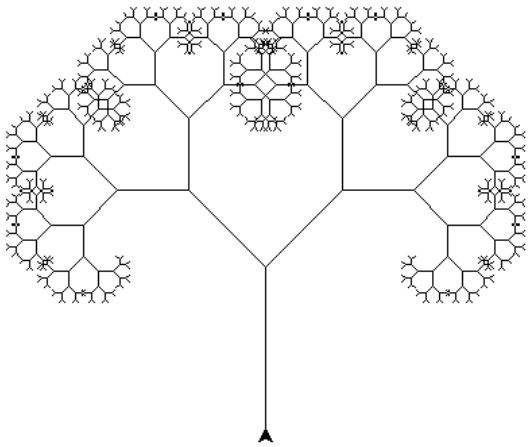


Figura 1. Árbol fractal.

Con este resultado se podría decir que se cumplió con el objetivo de construir un árbol fractal simple. Pero, ¿qué tal si se construye un árbol que tenga una apariencia más similar a la de un árbol real?

Hojas

La primera mejora que se puede hacer al árbol para hacerlo más real es agregarle hojas. El dibujo de las hojas debe darse sólo en las ramas que no se dividen más, cuando no se hacen más llamadas recursivas. Esto es precisamente en el caso base de la función recursiva del árbol. Es aquí donde deben ubicarse los comandos para dibujar la hoja.

```
    def hoja(t, a):
        turtle.begin_fill()
        turtle.right(a/2)
        turtle.circle(t, a)
        turtle.left(180-a)
        turtle.circle(t, a)
        turtle.left(180-a/2)
        turtle.end_fill()
```

Para eso se utiliza una función que se encarga de dibujar dos segmentos de círculo iguales unidos por sus extremos formando ángulos que serán las puntas de la hoja. Esta función será invocada desde el caso base de la función del árbol para crear hojas en cada una de las ramas finales.

Figura 2. Hoja del árbol.

La función recibe como parámetro el valor `t`, que indica el tamaño del radio de los segmentos de círculo a dibujar, y el parámetro `a` indica el ángulo en el que los dos segmentos se unen. Entre más cerrado sea el ángulo de las puntas, más alargada va a parecer la hoja. Las instrucciones `begin_fill()` y `end_fill()` indican a la tortuga que cualquier figura cerrada que se dibuje entre esas dos instrucciones debe rellenarse con un color sólido.

Al poderse manipular la forma de la hoja a través de los parámetros de la función, es posible cambiar su forma cada vez que se dibuja el árbol para darle variedad al resultado. Esta es la siguiente modificación que se le hará al árbol, agregarle aleatoriedad en sus valores para que el resultado no sea tan simétrico y tenga una apariencia más natural.

Esto se logrará de dos maneras:

1. Alterando su estructura (ángulo de las ramas y relación de tamaño entre cada rama y sus sub-ramas).
2. Alterando sus colores (agregar pequeñas variaciones de color entre los segmentos del tronco y entre las hojas).

Variaciones en la estructura

Existen varios aspectos de la estructura del árbol que se pueden modificar para darle un aspecto más realista. Uno de ellos es el ángulo en que nacen las sub-ramas a partir de una rama principal. En el árbol desarrollado hasta ahora, las ramas nacen en un ángulo de 45 grados, lo que da como resultado un árbol de apariencia muy simétrica y hasta en cierto modo "cuadrado". Esto se puede modificar para que el ángulo utilizado sea variable dentro del mismo árbol.

Otro aspecto que puede ayudar al árbol a verse más real es la relación de tamaño que existe entre cada rama y sus sub-ramas. En árbol generado hasta ahora, cada sub-rama mide 2/3 del tamaño de la rama de donde proviene, haciendo que todas las ramas tengan exactamente el mismo tamaño dependiendo del nivel de profundidad donde se encuentren. Es posible agregar variación en esta relación, de modo que de una misma rama nazcan sub-ramas de diferente tamaño, generando ramas más altas que otras, y más variedad dentro del árbol.

Adicionalmente, algunos detalles que terminan de afinar la figura del árbol son el grosor del tronco y la forma de las hojas. El grosor del tronco en principio debe ser grueso para las ramas iniciales y debe irse

haciendo más delgado conforme se va acercando a las hojas del árbol. Las hojas por su parte pueden tener variedad en su forma para que sean diferentes cada vez que se genera un árbol.

```
ANG = 20
RAND = 10
REL = 2/3
RANDT = 60
GROSORTRONCO = 2
TAMINIC = 150
TAMHOJA = 4
ANGHOJA = 180
PROF = 10
```

Se utilizarán una serie de variables globales con valores que se utilizarán como constantes a la hora de dibujar el árbol. ANG indica el ángulo base de inclinación de las ramas del árbol. Para que este ángulo no sea igual para todas las ramas se declara la variable RAND, que define el límite de variación (aleatoria) que se le sumará o restará al ángulo base para generar el ángulo de cada rama. Si ANG=20 y RAND=10 significa que la inclinación de cada rama será de 20 con una modificación aleatoria de -10 a +10 grados. Es decir, el ángulo de cada rama será un valor aleatorio entre 10 y 30 grados.

La variable REL indica la relación de tamaño entre cada rama y sus sub-ramas, y la variable RANDT indica el porcentaje de variación aleatorio que se le va a aplicar a la rama. Si RANDT=60 entonces el tamaño de cada rama puede variar un 60 % del tamaño original, ya sea más pequeño o más grande. Al tamaño específico de cada rama se le aplica la proporción en REL para indicar el tamaño base de las sub-ramas.

La variable GROSORTRONCO es una constante que indica cuántos pixeles de grosor debe sumársele al dibujo en general, esto para tener la posibilidad de dibujar árboles de tronco grueso como de tronco delgado. TAMINIC indica el tamaño en pixeles del segmento principal del árbol.

Las variables TAMHOJA y ANGHOJA en conjunto definen la apariencia de las hojas del árbol, indicando el tamaño de la hoja y del ángulo de las puntas respectivamente.

Por último la variable PROF indica cuántos niveles de profundidad se van a utilizar en la construcción del árbol.

```
def arbol(t, d):
    if d==0:
        turtle.forward(t)
        hoja(TAMHOJA, ANGHOJA)
        turtle.penup()
        turtle.back(t)
        turtle.pendown()
        return
    else:
        angulo1 = ANG + random.randrange(-RAND, RAND+1)
        angulo2 = ANG + random.randrange(-RAND, RAND+1)
        tamano = t + t*random.randrange(-RANDT, RANDT+1)/100
```

```
            turtle.pensize(d+GROSORTRONCO)
            turtle.forward(tamano)
            turtle.left(angulo1)
            arbol(t*REL, d-1)
            turtle.right(angulo1+angulo2)
            arbol(t*REL, d-1)
            turtle.left(angulo2)
            turtle.penup()
            turtle.back(tamano)
            turtle.pendown()
```

Para implementar los cambios sugeridos a la estructura del árbol se hacen algunas modificaciones a la función principal de dibujar el árbol. El caso base de la función ahora se encarga de dibujar un tallo y la hoja al final del mismo invocando a la función de dibujar hoja con los parámetros definidos en las variables globales de tamaño y ángulo.

La modificación sustancial de esta función se encuentra en el caso general. Como el propósito es variar los ángulos en que se dibujan las ramas de manera aleatoria, entonces se declaran dos variables `angulo1` y `angulo2` que contienen el ángulo respectivo de cada rama. El valor de cada una de estas variables se define a partir del valor ANG que es el ángulo base de las ramas y se le suma un valor aleatorio que puede ir desde −RAND hasta RAND. Para esto se utiliza la función `randrange` de la biblioteca `random`.

También se calcula el tamaño particular de la rama actual. El valor base para este tamaño es el valor `t` enviado como parámetro a la función, pero se le suma (o resta) un porcentaje aleatorio según el valor de variación que se encuentre en la variable RAND. De esta forma el tamaño resultante va a ser parecido al valor de `t`, pero aumentado o disminuido en un porcentaje.

El método `turtle.pensize` establece el ancho de la línea que dibuja la tortuga. Para darle grosor al tronco según el nivel de profundidad se aprovecha el valor del parámetro `d`, que es la profundidad restante de la rama que se dibuja actualmente. De esta forma el tronco dibujado es más grueso al inicio y más delgado conforme se acerca a la profundidad cero. A este valor se le suma la variable GROSORTRONCO para determinar el grosor general del árbol dibujado.

El resto del código de la función permanece sin cambios significativos. Luego de realizar estos cambios, y con los valores de las variables globales mostrados, se obtiene un resultado como el de la figura 3 al ejecutar la nueva versión de la función para dibujar el árbol.

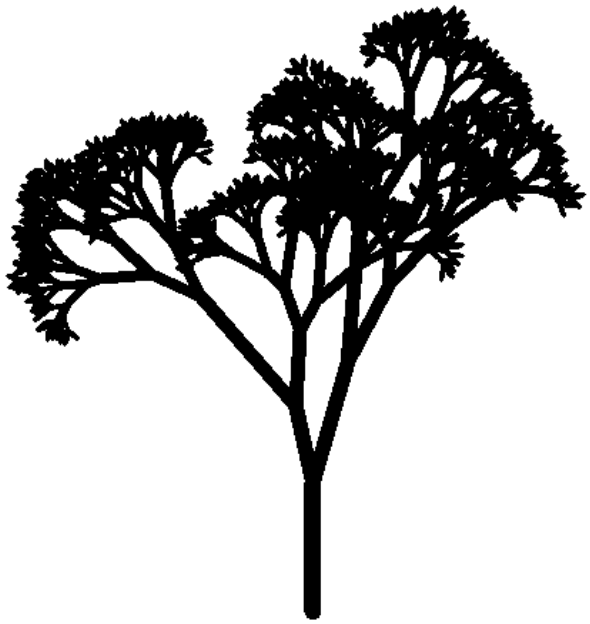


Figura 3. Árbol fractal con variaciones en los ángulos de las ramas.

Variaciones en los colores

A continuación se mostrarán algunas modificaciones que sirven para darle color al árbol. Una de las opciones más básicas es utilizar un color para dibujar el tronco y otro diferente para las hojas. Para hacer el dibujo más complejo se le agregarán pequeñas variaciones sobre esos colores para dibujar cada segmento del tronco y cada hoja del árbol, de modo que el resultado no sea un objeto de un solo color plano, si no de muchos colores parecidos según una serie de valores predeterminados.

Pero antes de entrar en detalle es importante mencionar que `turtle` utiliza constantes en cadenas para definir los colores a utilizar en las líneas y en los rellenos, entonces por ejemplo `turtle.color("blue")` hace que la tortuga dibuje líneas y rellenos con color azul. Trabajar con nombres de colores en cadenas de texto se hace un poco complicado, por lo que en este ejemplo se cambiará a modo RGB para especificar colores por medio del comando `turtle.colormode(255)`.

En el modo RGB cada color está dado por una tupla de tres valores enteros entre 0 y 255. El primer valor corresponde a la cantidad de rojo, el segundo al verde y el tercero a la cantidad de azul utilizado para generar el color. Se recomienda consultar sobre este formato de especificación de colores si no se está familiarizado con él [1].

```
    CTRONCO = (100,80,0)
    CTRONCOVAR = 30
    CHOJAS =  (66,190,9)
    CHOJASVAR = 100
    CFONDO = (255,255,255)
```

Para llevar a cabo la variación de los colores dentro del árbol se utilizarán algunas variables globales. La variable `CTRONCO` es una tupla RGB con el color base que se utilizará en el tronco. La variable

CTRONCOVAR indica la cantidad de variación que se le hará a los tres valores enteros de la tupla en CTRONCO. Si CTRONCOVAR=40, entonces el color de cada segmento del árbol estará formado por una variación de hasta 40 unidades (positivo o negativo) del color especificado en CTRONCO. Si se agrega mucha variación, entonces los colores resultantes pueden ser muy diferentes, pero si el valor de variación es bajo, entonces los colores van ser diferentes pero parecidos.

Bajo la misma forma de trabajo se declaran las variables CHOJAS y CHOJASVAR, que definen el color base de las hojas del árbol como la cantidad de variación permitida en los colores de las mismas. Finalmente la variable CFONDO simplemente guarda el color de fondo del dibujo.

Dada la estrategia de tener un color base y generar una variación del mismo para ir pintando las diferentes partes del árbol, se hace necesaria una función que lleve a cabo esta variación.

```
    def variacioncolor(color, var):
        Rd = random.randrange(-var, var+1)
        Gd = random.randrange(-var, var+1)
        Bd = random.randrange(-var, var+1)
        R, G, B = color
        R += Rd
        G += Gd
        B += Bd
        if R > 255:
            R = 255
        elif R < 0:
            R = 0
        if G > 255:
            G = 255
        elif G < 0:
            G = 0
        if B > 255:
            B = 255
        elif B < 0:
            B = 0
        return R, G, B
```

Esta función recibe el parámetro color que es una tupla de tres valores enteros (un color en RGB) y un parámetro var que es la variación máxima permitida. La función genera valores aleatorios entre −var y var para alterar el color original. Se tiene el cuidado de no permitir resultados que sean negativos ni mayores que 255, ya que no son permitidos por el modo RGB. Retorna una tupla con el nuevo color generado.

Con esta función es posible utilizar una llamada para alterar el color con el que la tortuga está dibujando actualmente. Esto se agrega en todos los puntos del código donde se tiene que cambiar de color, es decir, al inicio de cada segmento del tronco y al dibujar cada hoja.

```
    turtle.color(variacioncolor(CHOJAS, CHOJASVAR))
```

Al agregar estas variaciones en las funciones de árbol y hoja se puede obtener un resultado como el mostrado en la figura 4, donde cada segmento y cada hoja muestra un color particular.

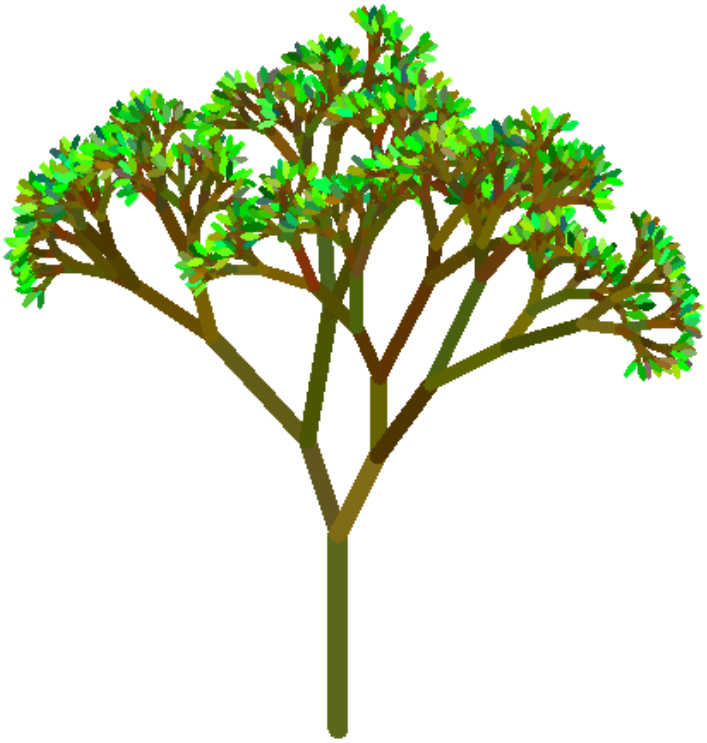


Figura 4. Árbol fractal con variaciones en los ángulos y colores de las ramas y hojas.

Algunas pruebas

Dado que los valores utilizados para definir la estructura y color del árbol se encuentran parametrizados en las variables globales, muy fácil modificarlas para ver qué resultados se obtienen. Por ejemplo, al utilizar un mayor valor en el ángulo de las ramas (ANG) se obtienen árboles más abiertos como el de la figura 5.

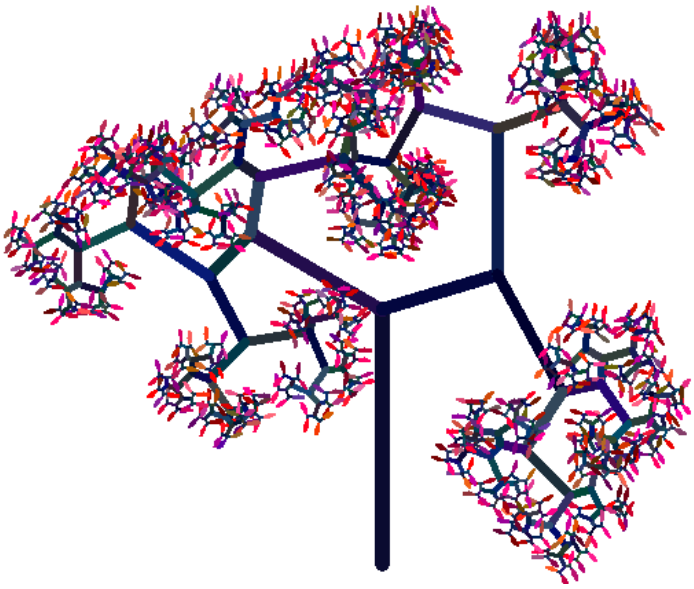


Figura 5. Árbol fractal con ángulos más abiertos.

Por otro lado, si se utiliza REL=4/5 con un valor más cercano a uno, pero con ANG=30, entonces el resultado es un poco más intrincado y con más detalle ya que es posible apreciar las ramas más pequeñas del árbol.

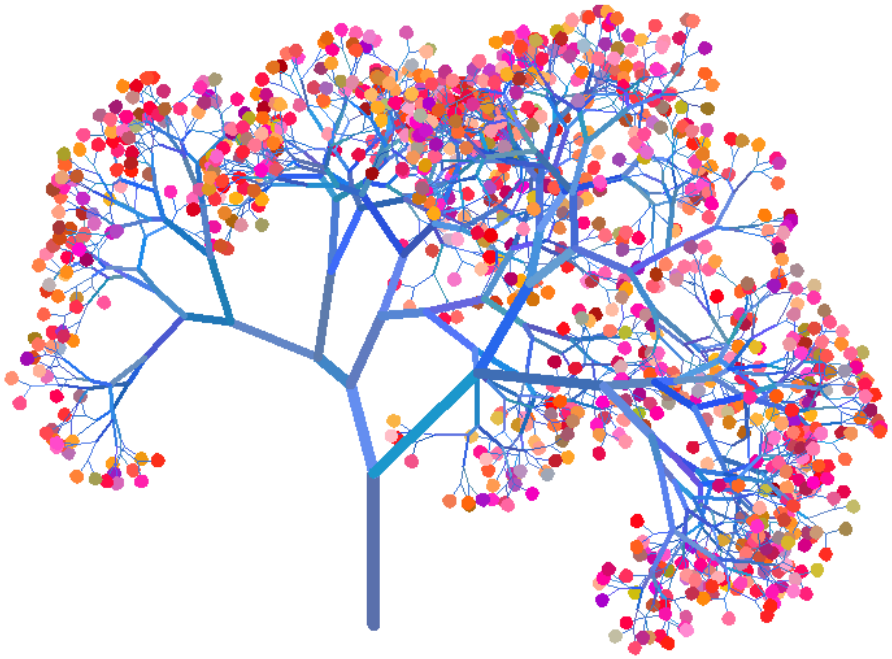


Figura 6. Árbol fractal con variación en la longitud de las ramas.

Una modificación adicional que puede hacerse al código es hacer una función que determine de forma aleatoria los valores iniciales de las variables globales, para así obtener múltiples variaciones de los árboles sin tener que modificar el código fuente.

Código fuente

```
    import turtle
    import random

    #Variables globales
    ANG = 30    #ángulo de inclinación para las ramas
    RAND = 30   #factor de aleatoriedad del ángulo de inclinación (grados)
    REL = 4/5   #relación entre la rama y las sub ramas
    RANDT = 60  #factor de aleatoriedad en el tamaño de las ramas (%)
    GROSORTRONCO = 0        #pixeles que se le suman al grosor del árbol
    TAMINIC = 100           #tamaño del tronco inicial en pixeles
    TAMHOJA = 5    #tamaño de la hoja
    ANGHOJA = 180    #ángulo de las puntas de las hojas (180 = círculos)
    PROF = 10       #cantidad de niveles en el árbol (más de 10 puede durar
mucho dibujándose)

    CTRONCO = (67,120,211) #color del tronco. tres números entre 0 y 255
    CTRONCOVAR = 40         #factor de aleatoriedad en el color del tronco
    CHOJAS =  (255,90,109)    #color de las hojas
    CHOJASVAR = 100         #factor de aleatoriedad en el color de las hojas
    CFONDO = (255,255,255)  #color de fondo

    # Función que dibuja un árbol fractal
    # Entradas:
    #   t: tamaño del segmento inicial en pixeles
    #   d: profundidad total del árbol
    # Salidas:
    #   Dibujo del árbol en pantalla
    # Restricciones: no
    def arbol(t, d):
        if d==0:
            turtle.forward(t)
            hoja(TAMHOJA, ANGHOJA)
            turtle.penup()
            turtle.back(t)
            turtle.pendown()
            turtle.color(CTRONCO)
            return
        else:
            angulo1 = ANG + random.randrange(-RAND, RAND+1)
            angulo2 = ANG + random.randrange(-RAND, RAND+1)
            tamano = t + t*random.randrange(-RANDT, RANDT+1)/100
            colortronco = variacioncolor(CTRONCO, CTRONCOVAR)
            turtle.color(colortronco)
            turtle.pensize(d+GROSORTRONCO)
```

```
            turtle.forward(tamano)
            turtle.left(angulo1)
            arbol(t*REL, d-1)
            turtle.right(angulo1+angulo2)
            arbol(t*REL, d-1)
            turtle.color(colortronco)
            turtle.left(angulo2)
            turtle.penup()
            turtle.back(tamano)
            turtle.pendown()

    # Función que dibuja una hoja
    # Entradas:
    #   t: tamaño de la hoja
    #   a: ángulo de las puntas de las hojas
    # Salidas:
    #   Dibujo de una hoja en la posición actual de la tortuga
    # Restricciones: no
    def hoja(t, a):
        turtle.color(variacioncolor(CHOJAS, CHOJASVAR))
        turtle.begin_fill()
        turtle.right(a/2)
        turtle.circle(t, a)
        turtle.left(180-a)
        turtle.circle(t, a)
        turtle.left(180-a/2)
        turtle.end_fill()

    # Función que genera una variación de un color en RGB
    # Entradas:
    #   color: tupla con tres valores enteros entre 0 y 255
    #   var: cantidad máxima de variación permitida en los valores RGB
    # Salidas:
    #   Tupla de tres valores enteros entre 0 y 255 que es una variación
    #   del color original.
    # Restricciones: no
    def variacioncolor(color, var):
        Rd = random.randrange(-var, var+1)
        Gd = random.randrange(-var, var+1)
        Bd = random.randrange(-var, var+1)
        R, G, B = color
        R += Rd
        G += Gd
        B += Bd
        if R > 255:
            R = 255
        elif R < 0:
            R = 0
        if G > 255:
            G = 255
```

```
        elif G < 0:
            G = 0
        if B > 255:
            B = 255
        elif B < 0:
            B = 0
        return R, G, B

    # Función que inicializa la posición de la tortuga e invoca a la función
    # de dibujar árbol fractal.
    def init():
        turtle.speed(0)
        turtle.colormode(255)
        turtle.clear()
        turtle.penup()
        turtle.home()
        turtle.left(90)
        turtle.back(200)
        turtle.pendown()
        turtle.hideturtle()
        turtle.color(CTRONCO)
        turtle.bgcolor(CFONDO)
        arbol(TAMINIC, PROF)
        turtle.done()

    init()
```

Referencias

[1] Fractal. (2013, May 24). In *Wikipedia, The Free Encyclopedia*. Retrieved 19:22, May 27, 2013, from http://en.wikipedia.org/w/index.php?title=Fractal&oldid=556618846

[2] Turtle graphics. (2013, March 21). In *Wikipedia, The Free Encyclopedia*. Retrieved 19:46, May 27, 2013, from http://en.wikipedia.org/w/index.php?title=Turtle_graphics&oldid=545963139

[3] RGB color model. (2013, April 29). In *Wikipedia, The Free Encyclopedia*. Retrieved 20:27, May 28, 2013, from http://en.wikipedia.org/w/index.php?title=RGB_color_model&oldid=552766670

[4] Adamchik, V. S. (n.d.). Binary Trees. School of Computer Cience, Carnegie Mellon. Retrieved May 25, 2013, from http://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html

Mauricio Avilés es bachiller en Ingeniería en Computación del Instituto Tecnológico de Costa Rica, máster en Educación con énfasis en Docencia. Es profesor de los cursos de Introducción a la Programación, Taller de Introducción a la Programación, Estructuras de Datos y Programación Orientada a Objetos en la Escuela de Computación del Instituto Tecnológico de Costa Rica.