CSCI 134 Fall 2021: Graphical Recursion

October 29, 2021

Shikha Singh, 9AM
Jeannie Albrecht, 10AM
Announcements & Logistics

- **Lab 5** will be feedback released today
- **Lab 7** has been posted: focuses on recursion!
- **HW 6** due Monday, 10 pm: covers lambda sorting, dictionaries, sets
- Midterms have been graded: will be returned at the end of class
  - Last ten minutes of lecture: discuss midterm feedback
- **Scheduled final:** Sat, December 18, 9:30 am, details TBD
- **CS TA applications** due today; **CS preregistration colloquium** today at 2.35 pm in Wege (TCL 123)
- **Monday, Nov 1:** No class! Watch video(s) instead!
  - Videos will be very helpful for lab

Do You Have Any Questions?
Last Time

• Discussed more examples with recursion & recursive approach to problem solving

• Breaking problems down into smaller pieces is crucial in CS (called decomposition or divide and conquer)
  • Some of the recursion we do in this class is meant to help you understand the concept and is not necessarily the most efficient way to solve the problem.
  • This problem solving technique is very important in CS: comes up in CS 136, and CS 256 (memoization with recursion leads to very efficient algorithms)

• Introduced turtle module and basic commands (briefly)
Today’s Plan

• More practice with using the turtle
• Graphical recursion examples
• Understanding function invariance and why it matters when doing recursion
Review: Turtle

- Python has a built-in module named `turtle`. See the Python turtle module API for details.
- Basic turtle commands:

  Use `from turtle import *` to use these commands:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd(dist)</code></td>
<td>turtle moves forward by <code>dist</code></td>
</tr>
<tr>
<td><code>bk(dist)</code></td>
<td>turtle moves backward by <code>dist</code></td>
</tr>
<tr>
<td><code>lt(angle)</code></td>
<td>turtle turns left <code>angle</code> degrees</td>
</tr>
<tr>
<td><code>rt(angle)</code></td>
<td>turtle turns right <code>angle</code> degrees</td>
</tr>
<tr>
<td><code>pu()</code></td>
<td>(pen up) turtle raises pen in belly</td>
</tr>
<tr>
<td><code>pd()</code></td>
<td>(pen down) turtle lower pen in belly</td>
</tr>
<tr>
<td><code>pensize(width)</code></td>
<td>sets the thickness of turtle's pen to <code>width</code></td>
</tr>
<tr>
<td><code>pencolor(color)</code></td>
<td>sets the color of turtle's pen to <code>color</code></td>
</tr>
<tr>
<td><code>shape(shp)</code></td>
<td>sets the turtle's shape to <code>shp</code></td>
</tr>
<tr>
<td><code>home()</code></td>
<td>turtle returns to (0,0) (center of screen)</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>delete turtle drawings; no change to turtle's state</td>
</tr>
<tr>
<td><code>reset()</code></td>
<td>delete turtle drawings; reset turtle's state</td>
</tr>
<tr>
<td><code>setup(width,height)</code></td>
<td>create a turtle window of given <code>width</code> and <code>height</code></td>
</tr>
</tbody>
</table>
Review: Basic Turtle Movement

- forward or `fd`, left or `lt`, right or `rt`, backward or `back`

```python
In [1]: from turtle import *

#setup(400,400, 0, 0) # open the Turtle graphics window
fd(100) # move turtle forward 100 pixels

In [2]: lt(90) # change direction by 90 degrees left
fd(100) # move forward 100 pixels

In [3]: # completing the square
lt(90)
fd(100)

In [4]: lt(90)
fd(100)
```
Review: Drawing Polygons

- We can use loops to draw shapes, such as polygons

```python
def polygon(numSides, sideLength):
    """Draws a polygon with the specified number of sides, each with the specified length.""
    for side in range(numSides):
        fd(sideLength)
        lt(360/numSides)
```

![Polygons](image)
Example: Concentric circles. Write the following recursive function.

\[ \text{concentricCirc}(\text{radius}, \text{thickness}, \text{color}_1, \text{color}_2) \]

- \textbf{radius}: radius of the outermost circle
- \textbf{thickness}: thickness of the band between circles
- \textbf{color}_1: color of the outermost circle
- \textbf{color}_2: color that alternates with \text{color}_1
from turtle import *

def drawDisc(radius, color):
    """Draws a circle of given radius and color with centre (0,0) assuming turtle's initial position is (0, -radius)"""
    pd()
    pen(fillcolor = color)
    begin_fill()
    circle(radius)
    end_fill()
    pu()
Concentric Circles: Recursive Approach

• Identify smaller version of the problem
  • If we just draw outermost circle, can trust recursion to draw all the remaining inner circles

• How are the parameters of the function changing?
  • Radius getting smaller, by how much?
  • Colors are alternating

• **Base case**: when does the radius get small enough that we don't need to draw anything?
  • When radius is less than the thickness
Concentric Circles: Recursive Approach

• Identify smaller version of the problem
  • If we just draw outermost circle, can trust recursion to draw all the remaining inner circles

• How are the parameters of the function changing?
  • Radius getting smaller, by how much?
  • Colors are alternating

• **Base case**: when does the radius get small enough that we don't need to draw anything?

```python
def concentricCirc(radius, thickness, color1, color2):
    if radius < thickness:
        pass # do nothing
    else:
        pass
```
Concentric Circles: Recursive Approach

- **Recursive case:**
  - Take an action that contributes to the solution
  - Check that turtle is positioned at the right place
  - Make recursive calls on a smaller subproblem
  - Ensure the parameters of the function calls change appropriately

```python
def concentricCirc(radius, thickness, color1, color2):
    if radius < thickness:
        pass  # do nothing
    else:
        drawDisc(radius, color1)
        lt(90)
        fd(thickness)
        rt(90)
        concentricCirc(radius - thickness, thickness, color2, color1)
```
Invariance of Functions

• A function is **invariant** relative to an object’s state if the state of the object is the same *before* and *after* a function is invoked.

• Right now our *concentricCirc* function is not invariant with respect to the position of the turtle (the turtle does not end where it starts).

• How can we make it invariant, that is, return the turtle to starting position?

```python
def concentricCirc(radius, thickness, color1, color2):
    if radius < thickness:
        pass  # do nothing
    else:
        drawDisc(radius, color1)
        lt(90)
        fd(thickness)
        rt(90)
        concentricCirc(radius - thickness, thickness, color2, color1)
```
Invariant Concentric Circles

- Ensuring that we "undo" turtle movements that happened before the recursive call, after the recursive call results in invariance

```python
def concentricCirc(radius, thickness, color1, color2):
    if radius < thickness:
        pass  # do nothing
    else:
        drawDisc(radius, color1)
        lt(90)
        fd(thickness)
        rt(90)
        concentricCirc(radius - thickness, thickness, color2, color1)
        lt(90)
        bk(thickness)
        rt(90)
```
Invariance of Recursive Functions

• Why do we care about invariance?
  • It is a good property to have for recursive functions
  • Is not crucial to correctness when we have a single recursive call
  • However, with multiple recursive calls, our graphical functions will not work properly if they are not invariant

• Let's do an example with multiple recursive calls
  • Nested circles (see picture)
Multiple Recursive Call

• **Example:** Nested circles. Write the following recursive function

```python
nestedCircles(radius, minRadius, color1, color2)

  • `radius`: radius of the outermost circle
  • `minRadius`: minimum radius of any circle
  • `color1`: color of the outermost circle
  • `color2`: color that alternates with `color1`
```
Nested Circles

• **Base case?**
  
  • When radius becomes less than minRadius

```python
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        pass
```

• **Recursive case**
  
  • Draw the outer circle
  
  • Position turtle for recursive calls
  
  • How many recursive calls?
Nested Circles

- **Recursive case**
  - Draw the outer circle
  - Position turtle for right recursive subcircle: trust recursion fairy!

```python
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        # contribute to the solution
drawDisc(radius, color1)

        # position the turtle at the right place
lt(90); fd(radius/2); rt(90); fd(radius/2)

        # draw right subcircle recursively
nestedCircles(radius/2, minRadius, color2, color1)
```
Nested Circles

- **Recursive case**
  - Suppose right subcircle is drawn
  - If our function is invariant where can we assume the turtle is after returning from the recursive call?

```python
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        # contribute to the solution
drawDisc(radius, color1)

        # position the turtle at the right place
    lt(90); fd(radius/2); rt(90); fd(radius/2)

        # draw right subcircle recursively
    nestedCircles(radius/2, minRadius, color2, color1)
```
Nested Circles

- **Recursive case**
  - Now need to move the turtle to draw left subcircle recursively

```python
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        # contribute to the solution
drawDisc(radius, color1)
        # position the turtle at the right place
        lt(90); fd(radius/2); rt(90); fd(radius/2)
        # draw right subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
        # position turtle for left subcircle
        bk(radius)
        # draw left subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
```
Nested Circles

- **Recursive case**

  - Assuming invariance, where does the turtle end up after the call?

```python
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        # contribute to the solution
        drawDisc(radius, color1)
        # position the turtle at the right place
        lt(90); fd(radius/2); rt(90); fd(radius/2)
        # draw right subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
        # position turtle for left subcircle
        bk(radius)
        # draw left subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
```
Nested Circles

- **Recursive case**
  - What is the final thing we need to do?

```python
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        # contribute to the solution
        drawDisc(radius, color1)
        # position the turtle at the right place
        lt(90); fd(radius/2); rt(90); fd(radius/2)
        # draw right subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
        # position turtle for left subcircle
        bk(radius)
        # draw left subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
```
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        pass  # do nothing
    else:
        # contribute to the solution
        drawDisc(radius, color1)
        # position the turtle at the right place
        lt(90); fd(radius/2); rt(90); fd(radius/2)
        # draw right subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
        # position turtle for left subcircle
        bk(radius)
        # draw left subcircle recursively
        nestedCircles(radius/2, minRadius, color2, color1)
        # bring turtle back to start position
        fd(radius/2);  lt(90);  bk(radius/2);  rt(90)
Fruitful Version: Nested Circle

- Suppose we wanted to keep track of number of colors recursively

\[ \text{nestedCircles}(\text{radius, minRadius, color1, color2}) \]

- \text{radius}: radius of the outermost circle
- \text{minRadius}: minimum radius of any circle
- \text{color1}: color of the outermost circle
- \text{color2}: color that alternates with color1

Must return tuple of values:

- first item is \# of circles of \text{color1}
- second item is \# of circles of \text{color2}
Fruitful Version: Nested Circle

• How would this change the function?
• What should base case return?
• How many circles of each color are drawn directly by the main function (excluding any recursive calls)?
• How do we keep track of number of circles of each color drawn by recursive calls?
  • Wishful thinking/recursion fairy!
def nestedCircles(radius, minRadius, color1=lgreen, color2=dgreen):
    if radius < minRadius:
        # draw nothing
        return (0, 0)
    else:
        # contribute to the solution
drawDisc(radius, color1)
        # position the turtle at the right place
        lt(90); fd(radius/2); rt(90); fd(radius/2)
        # draw right subcircle recursively
        r2, r1 = nestedCircles(radius/2, minRadius, color2, color1)
        # position turtle for left subcircle
        bk(radius)
        # draw left subcircle recursively
        l2, l1 = nestedCircles(radius/2, minRadius, color2, color1)
        # bring turtle back to start position
        fd(radius/2); lt(90); bk(radius/2); rt(90)
        # drew 1 circle of color 1, zero of color 2
        return 1 + r1 + l1, r2 + l2
More Examples

• We will post more examples of graphical recursion on GLOW
• No in-person lecture on Monday: see videos instead to prepare for lab
Acknowledgments

These slides have been inspired from:

- http://cs111.wellesley.edu/spring19