Introduction to Programming

Lecture 4: recursion and classes

Ben Ruijl

Nikhef Amsterdam and Leiden University

November 1, 2016

Recursion

- Sometimes it is useful for a function to call itself with different arguments
- We call this recursion
- In maths it is common to express functions in terms of previous functions:

$$F(1) = 1$$
$$F(n) = n \cdot F(n-1)$$

What is F(4)?

Factorial

The previous function is called a factorial:

$$x! \equiv \prod_{i=1}^{x} i = x \cdot (x-1) \cdot (x-2) \cdots 1$$

In code:

```
1 int factorial(int x) {
2     int result = 1;
3     for (int i = 1; i <= x; i++) {
4         result *= i;
5     }
6     return result;
7 }
```

Recursively

$$F(1) = 1$$
$$F(n) = n \cdot F(n-1)$$

In code:

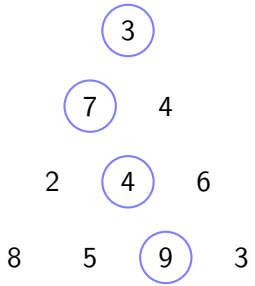
```
1 int factorial(int x) {
2     if (x == 1) return 1;
3     return x * factorial(x - 1);
4 }
```

The triangle problem

- Walking down, find path with highest score:

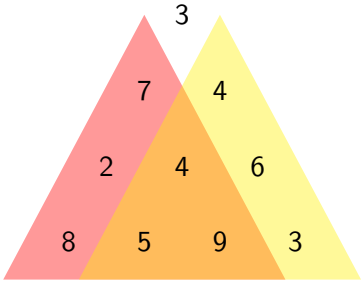
The triangle problem

- Walking down, find path with highest score:



Recursive solution

The problem repeats itself!

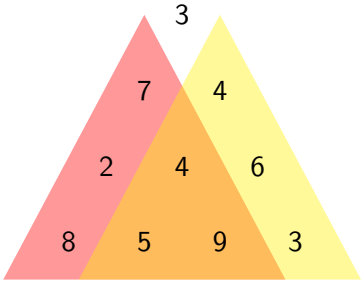


Recursive solution

The problem repeats itself!

Recursive strategy:

- Value at 3?
- Calculate value at 7
- Calculate value at 4
- Take the maximum



Coin problem

- The euro has the following coins: 1, 2, 5, 10, 20, and 50 cents and 1 euro and 2 euro
- In how many ways can you make 2 euros?
- Some examples:
- $2 \cdot 100$
- $3 \cdot 50 + 2 \cdot 20 + 2 \cdot 5$
- ...

Coin problem

- Say we pick a 50 cent coin
- We now have a similar problem of finding the number of ways to make 1.50
- We can solve this recursion:
  - Start with the largest coins first
  - We go through all the possible coins we can take
  - Take the coin and solve the subproblem of 2 minus the coin
- If m is the amount to obtain, and we use coins k or less, we have:

$$count(m, 1) = 1$$
$$count(m, k \text{ or less}) = \sum_{c \leq k} count(m - c, c)$$

Coin problem

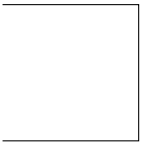
```
1  int[] coins = {200, 100, 50, 20, 10, 5, 2, 1};
2
3  int count(int money, int maxcoin) {
4      int sum = 0;
5      for(int i = maxcoin; i < coins.length; i++) {
6          if (money == coins[i]) sum++;
7          if (money - coins[i] > 0)
8            sum += count(money - coins[i], i);
9      }
10     return sum;
11 }
12
13 void setup() {  println(count(200, 0)); }
```

Fractals

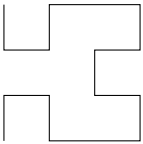
A fractal is a pattern that repeats itself indefinitely:

Figure: Fractal broccoli

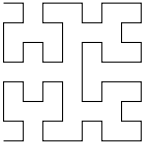
Hilbert curve



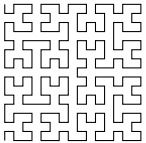
(a) $n = 1$



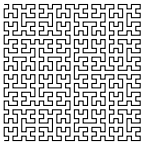
(b) $n = 2$



(c) $n = 3$



(d) $n = 4$



(e) $n = 5$

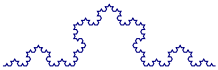
Koch curve

(a) $n = 0$

(b) $n = 1$

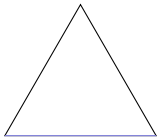
(c) $n = 2$

(d) $n = 3$

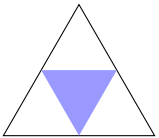


(e) $n = 4$

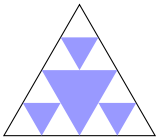
Sierpinski triangle



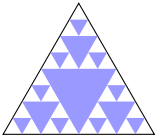
(a) n=1



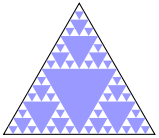
(b) n=2



(c) n=3



(d) n=4



(e) n=5

Tree fractal

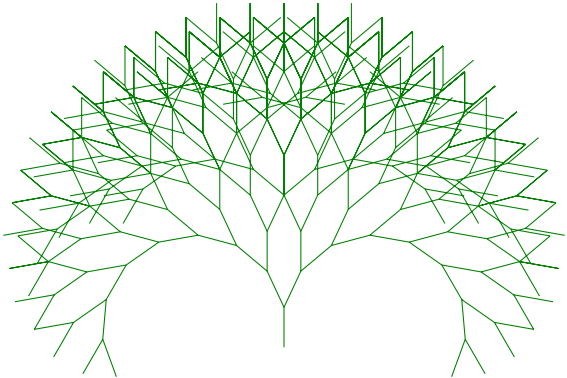


Figure: A fractal tree

Tree fractal

What is the component that is repeated?

Figure: A fractal tree

Drawing

For every branch we draw, we draw two more: one to the left and one to the right

Tree fractal

- This is recursion, we keep drawing the same figure!
- The position and the angle differ each time
- Pseudo-code:

```
1 // draw a single branch
2 function drawTree(x, y, angle) {
3   newx = x + cos(angle) * 10;
4   newy = y + sin(angle) * 10;
5   line (x, y, newx, newy);
6 }
```

Adding recursion

Keep making new branches:

```
function drawTree(x, y, angle) {
  newx = x + cos(angle) * 10;
  newy = y + sin(angle) * 10;
  line (x, y, newx, newy);
  drawTree(newx, newy, angle - 25); // to the left
  drawTree(newx, newy, angle + 25); // to the right
}
```

This code will never stop!

Maximum depth

Add a maximum depth:

```
1 function drawTree(x, y, angle, depth) {
2   if (depth == 0) return; // stop
3   newx = x + cos(angle) * 10;
4   newy = y + sin(angle) * 10;
5   line (x, y, newx, newy);
6   drawTree(newx, newy, angle - 25, depth - 1);
7   drawTree(newx, newy, angle + 25, depth - 1);
8 }
```

Tree implementation

```
1  void drawTree(float x, float y, float a, int d) {
2    if (d == 0) return;
3    float x2 = x + cos(a) * 10.0;
4    float y2 = y + sin(a) * 10.0;
5    line(x, y, x2, y2);
6    drawTree(x2, y2, a - 20*PI/180, d - 1);
7    drawTree(x2, y2, a + 20*PI/180, d - 1);
8  }
9
10 void draw() {
11   background(255);
12   drawTree(320, 600, -PI/2, 9);
13 }
```

Classes

- Say you want to make 20 spaceships with a position, health, and velocity
- We have to keep track of all these variables in arrays:

```
1 float[] x = new float[20];
2 float[] y = new float[20];
3 float[] health = new float[20];
4 float[] vx = new float[20];
5 float[] vy = new float[20];
```

- This is really cumbersome
- You can group together code that belongs together in a class

Classes

Create a class that has all the information:

```
1 class Spaceship {
2   float x, y, health, vx, vy;
3 }
4
5 void setup() {
6   Spaceship player = new Spaceship();
7   player.health = 100;
8   player.x = 40;
9 }
```

Access elements with a dot

Class methods (functions)

```
1  class Spaceship {
2    float health;
3
4    void takeDamage(float damage) {
5      health -= damage;
6    }
7  }
8
9  void setup() {
10   Spaceship a = new Spaceship;
11   a.takeDamage(10);
12   println(a.health);
13 }
```

Constructors

A constructor sets initial values for a class instance:

```
1  class Spaceship {
2    float x, y, health;
3
4    Spaceship(float x, float y) {
5      this.x = x;
6      this.y = y;
7      health = 100;
8    }
9  }
10 void setup() {
11   Spaceship a = new Spaceship(100, 100);
12 }
```

Arrays of classes

```
1  class Spaceship {
2    Spaceship(float x, float y) {
3      //...
4    }
5  }
6
7  void setup() {
8    Spaceship[] a = new Spaceship[100];
9
10   for(int i = 0; i < 100; i++) {
11     a[i] = new Spaceship(i * 100, 100);
12   }
13 }
```

Dynamic arrays

`ArrayList<yourtype>` is a class for dynamic arrays:

```
1  class Spaceship {
2    Spaceship(int x, int y) {
3    }
4  }
5
6  void setup() {
7    ArrayList<Spaceship> a = new ArrayList<Spaceship>();
8    a.add(new Spaceship(30, 20));
9    a.add(new Spaceship(30, 40));
10   println(a.size());
11 }
```

References

- You can read more about built-in classes in the manual
- For example: `String`, `PImage`, `ArrayList`
- More on building classes: `http://processing.org/tutorials/objects/`