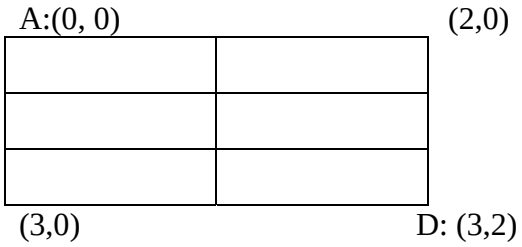
Lecture 27

Recursive Counting

Exercise 1: Paths in a grid [10 minutes]

Consider a 2D grid that looks as below:



You have to go from point A (0,0) to point D (3,2). You can make only two moves:

1) Travel one step right
2) Travel one step down

How many ways are there to from A to D ?

Answer: 10 paths

Lecture: Path Counting [15 minutes]

2D Grid Problem

More generally...
(curR, curC)

(maxR, maxC)

How many ways are there to get from (curR, curC) to (maxR, maxC) travelling only right or downward?

Main Idea

From every point, you have two choices.
(curR, curC)
----------> (curR, curC + 1)
|
|
|
\|/
(curR + 1, curC)

```
//Partial Solution
public static int countPaths(int curR, int curC, int maxR, int maxC)
	{

		int r = countPaths(curR, curC+1, maxR, maxC); //Going right
		int d = countPaths(curR+1, curC, maxR, maxC); //Going left
		return (r+d);
	 }




//Complete Solution
public static int countPaths(int curR, int curC, int maxR, int maxC)
	{
	if((curR>maxR)||(curC>maxC))
		return 0; //Base Case
	else if ((curR==maxR)&&(curC==maxC))
		return 1; //Base Case
	else
		{ //Recursive Case
		int r = countPaths(curR, curC+1, maxR, maxC); //Going right
		int d = countPaths(curR+1, curC, maxR, maxC); //Going left
		return (r+d);
		}
	}
```

Exercise 2: (Recursive Counting) [10 minutes]

You have a grid of points such as the one we discussed in lecture:
(0, 0) (0, 1) (0, 2) (0, 3) ..... (0, C)
(1, 0) (1, 1) ..
(2, 0) ... ...
...
(R, 0) (R, C)

The legal moves you are allowed to make are to move down three rows, to the right two rows, or diagonally downward and to the right.

For example, from (0, 0) you could move to (3,0), (0, 2), or (1, 1).

As in the lecture, if you reach the point (R, C), you are allowed to realize you are there, and end that path. As in lecture, you want your method to count the number of paths from any given point, to the destination, (R, C), making only legal moves along the way. Note that it is possible that you are at a location from which there are no legal moves to the destination point. You cannot use loops in this method.

Solution Discussion (10 minutes)

```
public static int countPaths(int curR, int curC, int R, int C)
{
// your code goes here
if ((curR > R) || (curC > C))
return 0;
else if ((curR == R) && (curC == C))
return 1;
else
        return countPaths(curR + 3, curC, R, C) +
        countPaths(curR, curC + 2, R, C) +
        countPaths(curR + 1, curC + 1, R, C);
}
```

Lecture (Pyramidal numbers) [20 minutes]

Exercise 3: Jumping stairs [15 minutes]

When walking up stairs, Jason noticed that sometimes he moves up **exactly one stair**, sometime he moves up **two stairs at once** (i.e. skipping a stair in between), and sometimes he moves up **three stairs at once** (i.e. skipping two stairs in between). He was curious how many different ways he could climb a set of stairs. You are to write a method **countWays** that takes an int numStairs as a parameter and returns an integer whose value is the number of different ways for Jason to climb the stairs. For example, if there are 3 stairs, Jason could either take one big step (skipping 2 stairs), take one medium step (skipping 1 stair) and one regular step, take one regular step and one medium step (skipping 1), or take 3 regular steps. So there are 4 different ways to climb 3 stairs. You cannot use loops in this method.

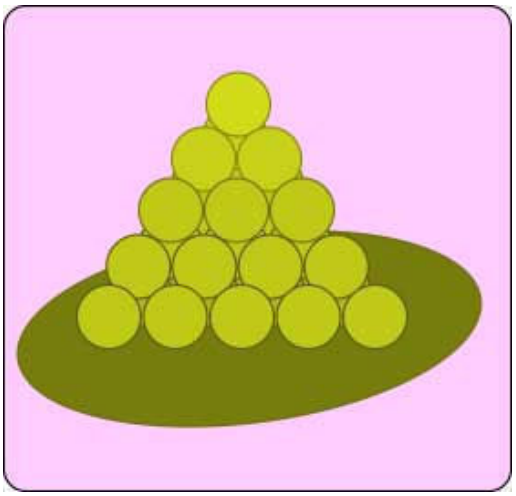
```
public static int countWays(int numStairs)
{
// your code goes here
if (numStairs < 0)
        return 0;
else if (numStairs == 0)
        return 1;
else
        return countWays(numStairs - 1) + countWays(numStairs - 2) +
        countWays(numStairs - 3);
}
```

Answer:

Three or four.

Pyramidal Numbers

Let us use three-sided pyramids (sometimes called tetrahedrons). To make the pyramid, start with a triangular arrangement of tennis balls for the bottom layer. Next stack up tennis balls layer at a time, as at right.



If the base of the pyramid has 5 balls along a side, how many balls does the entire pyramid have in it? Let us call this a **pyramidal number**.

Here is a chart that shows the number of balls in the side of a base, **N**, and the corresponding pyramidal numbers.

N	1	2	3	4	5	6	7
Pyramid(N)	1						

The picture starts with the base of the pyramid for **Pyramid(5)**. Click a few times to see the complete pyramid. Smaller pyramids can be found within this one. For small pyramids, count the balls within them. For larger pyramids, try to find a clever way to calculate the number of balls.

QUESTION 2:

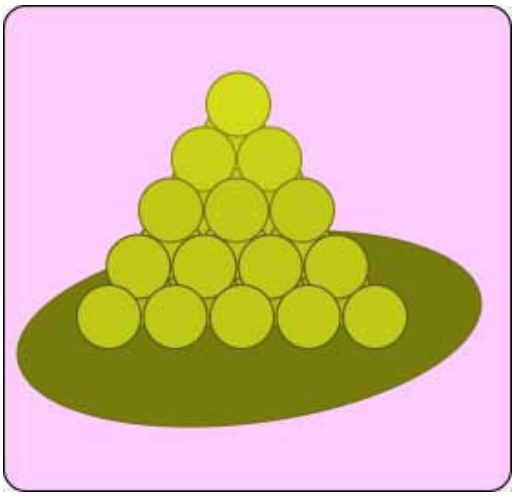
Click on the image a few times and fill in some of the chart.

Answer:

The completed, and expanded, chart is below.

Pyramid Scheme

In filling out the chart you might have noticed that it would be useful to know how many balls are in each layer of the pyramid. Of course, the number of the balls in the pyramid is the sum of the number of balls in each layer. But, the number of balls in layer N is `Triangle(N)`, the number of balls in a triangle with a side of N balls.



N	1	2	3	4	5	6	7
Triangle(N)	1	3	6	10	15	21	28
Pyramid(N)	1	4	10	20	35	56	84

Say that you know that there are `Pyramid(N-1)` balls in the first N-1 layers (counting from the top of the pyramid). Then the following scheme looks tempting:

```
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

QUESTION 3:

This looks suspiciously like a recursive definition. But what is missing?

Answer:

The base case of the definition.

Base Case of the Pyramid

A recursive definition (or a recursive algorithm) needs two parts:

1. If the problem is easy, solve it immediately.
2. If the problem can't be solved immediately, divide it into smaller problems, then:
   - Solve the smaller problems by applying this procedure to each of them.

In terms of pyramidal numbers, this is:

```
Pyramid(1) = 1
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

Oddly, the base case for pyramidal numbers is the peak of the pyramid.

QUESTION 4:

Oh no! This definition uses `Triangle(N)` without saying anything about it. Is this wrong?

Answer:

No.

Definition that uses another Function

As long as everything in the definition for `Pyramid()` is defined someplace, everything is fine. Here (for reference) is `Pyramid()`:

```
Pyramid(1) = 1
Pyramid(N) = Pyramid(N-1) + Triangle(N)
```

And here (for review) is `Triangle()`:

```
Triangle( 1 ) = 1
Triangle( N ) = N + Triangle( N-1 )
```

Given these two definitions (and, if you are picky, the definitions of addition and subtraction), `Pyramid()` is completely defined.

QUESTION 5:

Of course, given a definition, creating a Java method is just a matter of translation:

```
public int Pyramid( int N )
{
  if (          ==          )

    return          ;

  else

    return          (          ) +          (          );
}
```

Fill in the blanks. (You can easily do this with copy-and-paste from the definition of Pyramid.)

Answer:

```
public int Pyramid( int N )
{
  if ( N == 1 )

    return 1;

  else

    return Pyramid ( N-1 ) + Triangle ( N );
}
```

Pyramid Program

Here is a complete program that computes Pyramidal numbers. The user specifies N, the number of balls in a side of the base, in the command line.

```
class Calculate
{
  public int Triangle( int N )
  {
    if ( N == 1 )
      return 1;
    else
      return N + Triangle( N-1 );
  }

  public int Pyramid( int N )
  {
    if ( N == 1 )
      return 1;
    else
      return Pyramid ( N-1 ) + Triangle ( N );
  }
}

public class PyramidTester
{
  public static void main ( String[] args)
  {
    int argument = Integer.parseInt( args[0] ); // Get argument from the command line

    Calculate calc = new Calculate();
    int result = calc.Pyramid( argument );
    System.out.println("Pyramid(" + argument + ") is " + result );
  }
}
```

The program must include a method for `Triangle()`. Here is a run of the program:

```
C:\>java PyramidTester 12
Pyramid(12) is 364
```

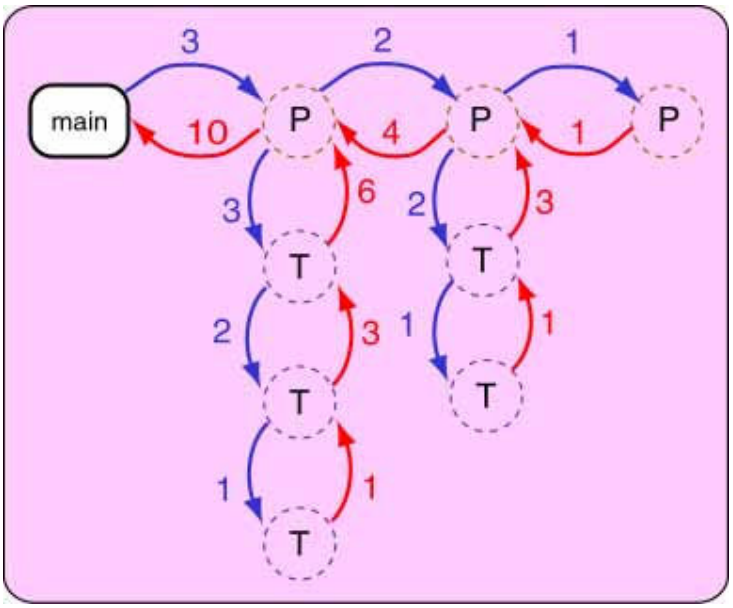
QUESTION 6:

Could the program (above) use an iterative method for `Triangle()` and a recursive method for `Pyramid()`?

Answer:

Sure. Either or both methods could be iterative or recursive.

Dynamic Pyramids



Here, for reference, are the two (recursive) methods:

```
public int Triangle( int N )
{
  if ( N == 1 )
    return 1;
  else
    return N + Triangle( N-1 );
}

public int Pyramid( int N )
{
  if ( N == 1 )
    return 1;
  else
    return Pyramid ( N-1 )
      + Triangle ( N );
}
```

The diagram shows the activation chain when the `main()` method has called

`Pyramid(3)`. This results in three activations of the `Pyramid()` method. Click on the diagram to see these activations.

When the base case `Pyramid(1)` is reached, it immediately returns a value of 1. Now the second activation, `Pyramid(2)`, calls `Triangle(2)` which results in activations of `Triangle()` added to the chain.

After two activations, `Triangle(2)`, returns a 3 to its caller. `Pyramid(2)` now has all the values it needs and returns a 4 to its caller.

`Pyramid(3)` now calls `Triangle(3)` which eventually returns a 6. The value 10 is returned to `main()`.

QUESTION 7:

It is equally valid to write `Pyramid()` with the methods in the last line reversed:

```
public int Pyramid( int N )
{
  if ( N == 1 )
    return 1;
  else
    return Triangle ( N ) + Pyramid ( N-1 ) ;  // Reversed Order
}
```

Will the sequence of activations in the activation chain be the same for this method?