MA2252 Introduction to Computing

Lecture 8

Part 2: Recursion

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

Learning outcomes

At the end of lecture, students will be able to

- understand recursion
- create recursive functions
- understand the difference between recursion and iteration
- solve recursion problems e.g. games

Recursion

Recursion occurs when something is defined in terms of itself.

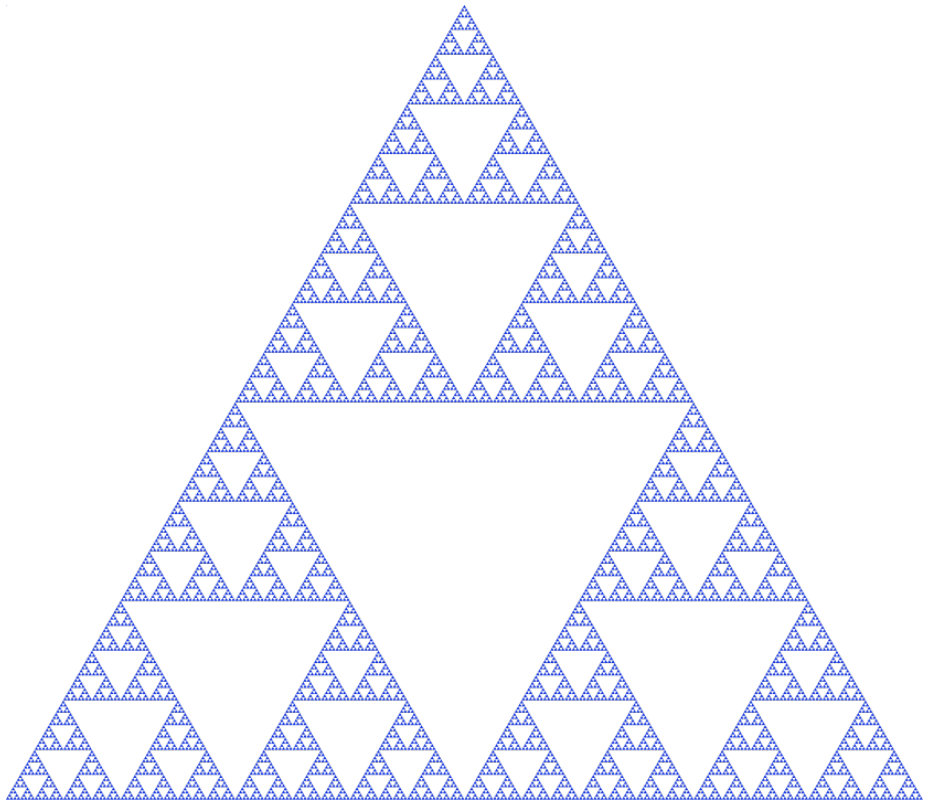
Examples:

- Romanesco broccoli[1]

example in biology

Recursion (contd.)

- Sierpiński triangle [2]



[1] Picture credit: Ivar Leidus
[2] Picture credit: Beojan Stanislaus

Recursive functions

A recursive function is defined in terms of itself.

Examples:

$0! = 1$

- Factorial function

$$n! = \begin{cases} 1, & n = 0 \\ n(n-1)!, & n \in \mathbb{N}. \end{cases}$$

base case

recursive step

$n = 1, 2, 3, \ldots$

$\sqrt{2+\sqrt{2+\sqrt{2+\cdots}}} = ?$

- $$f(n) = \begin{cases} \sqrt{2}, & n = 1 \\ \sqrt{2 + f(n-1)}, & n > 1. \end{cases}$$

$f(3) =$

$f(1) = \sqrt{2}$, $f(2) = \sqrt{2+f(1)} = \sqrt{2+\sqrt{2}}$

Recursive functions (contd.)

The definition of a recursive function includes

- **Base case:** Function's value is given or can be calculated without using recursion.
- **Recursive step:** Function's value is calculated by calling to itself.

For example, in factorial function's definition, the cases n=0 and n>0 are base case and recursive step respectively.

Recursive functions (contd.)

Example: Write a recursive function in MATLAB to find n!

```
function out = myfactorial(n)
if n==0
out=1;
else
out=n*myfactorial(n-1)
end
end
```

0! = 1

n = 3

i → n x (n-1)!

3! = 6

5! = 5x4x3x2x1 = 120

out = 3 * myfactorial(2)

= 3 * 2 * myfactorial(1)

= 3 x 2 x 1 x myfactorial(0)

= 1

= 6

Recursive functions (contd.)

Demo

Recursive functions (contd.)

The factorial function can also be calculated using while loop. Observe how tricky it is to code now!

```
function out = myfactorial_while(n)
if n==0||n==1
   out=1;
else
   fact=n;
   while n>1
      fact=fact*(n-1);
      n=n-1;
   end
   out=fact;
end
end
```

0! = 1 , 1! = 1 n > 1

n(n-1) = 2(2-1)

5! = 5 x 4 x 3 x 2 x 1

→ not a recursive function.

Recursion vs Iteration

e.g. using while loop was harder

Recursion	Iteration
Easier to code recursive functions	Harder to code recursive functions
creates extra workspace	uses only one workspace
consumes more memory	consumes less memory
code runs rather slow	code runs faster

Tower of Hanoi

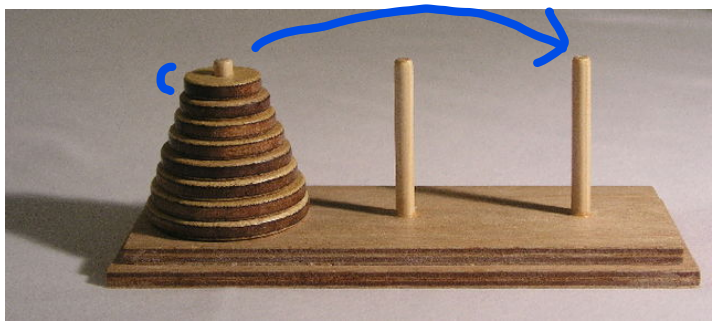


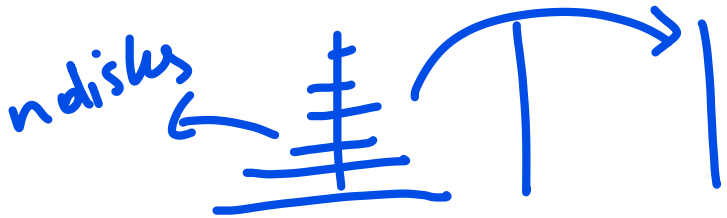
Figure: Tower of Hanoi game with 3 pegs [3]

Goal: To move an entire stack of disks from one peg to another.

Rules:

- Only one peg can be used at a time.
- Only top disk of a stack can be moved to an empty peg or top of other stack.
- Larger diameter peg cannot go on top of smaller diameter peg.

Recursion in Games (contd.)



Question:
What is the minimum number of moves required to move a stack of n disks to other peg?

steps

Possible strategy:
Exploit the recursive property of Tower of Hanoi game.

Recursion in Games (contd.)

The recursive definition of minimum number of moves function $f(n)$ is given as

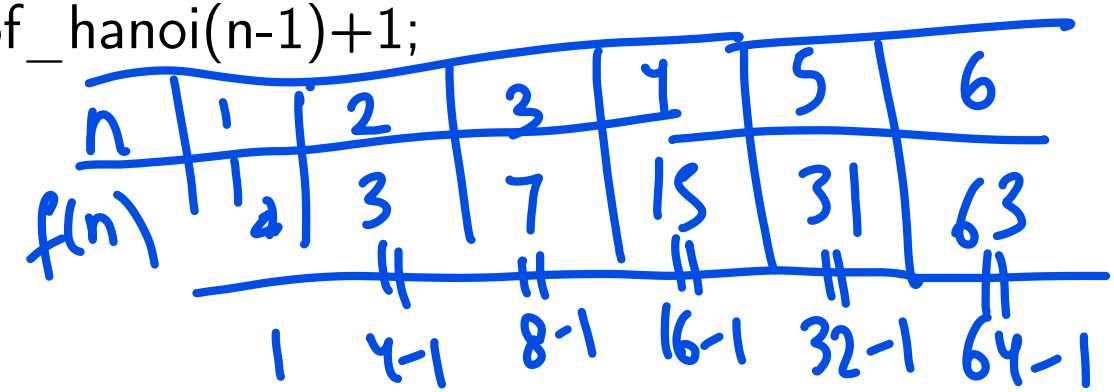
$$f(n) = \begin{cases} 1, & n = 1 \\ 2 * f(n-1) + 1, & n > 1. \end{cases}$$

Recursion in Games (contd.)

Write a recursive MATLAB function to find the minimum number of moves needed to play Tower of Hanoi game with n disks and 3 pegs.

```
function moves = tower_of_hanoi(n)
if n==1
   moves=1;
else
   moves=2*tower_of_hanoi(n-1)+1;
end
end
```

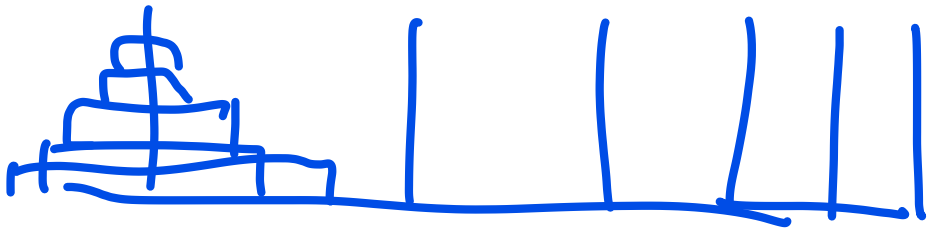
$f(n) = 2^n - 1$



Recursion in Games (contd.)

Demo

Recursion in Games (contd.)



Beyond 3 pegs?

beyond scope of this module

Frame-Stewart algorithm can be used to find minimum number of moves. The proof of optimality of this algorithm is still an unsolved problem in Mathematics.

Recreational mathematics solve puzzles or games

[3] Picture credit: Evanherk

End of Part 2

Please provide your feedback here