Seminar: Programming Language Concepts

Recursion

One of the classic programming problems that is often solved by recursion is the towers of Hanoi problem. A good explanation and walkthrough are provided by Cormen & Balkcom (n.d.) - the link is in the reading list. (the code they used for their visual example is provided on their website as well).

- Read the explanation, study the code and then create your own version using Python (if you want to make it more interesting you can use asterisks to represent the disks). Create a version that asks for the number of disks and then executes the moves, and then finally displays the number of moves executed.
- What is the (theoretical) maximum number of disks that your program can move without generating an error?
- What limits the number of iterations? What is the implication for application and system security?

```
# Recursive Python function to solve tower of hanoi
# n = Number of list to be moved
# from_stick = Stick where the disc is present
# to_stick = Stick where the disc is to be moved
# aux_stick = Auxilliary stick, the remaining one


def TowerOfHanoi(n, from_stick, to_stick, aux_stick):
    if n == 0:
        return
    TowerOfHanoi(n-1, from_stick, aux_stick, to_stick)
    print("Move disk", n, "from stick", from_stick, "to stick", to_stick)
    TowerOfHanoi(n-1, aux_stick, to_stick, from_stick)


# Driver code
n = 50

# A, C, B are the name of sticks
TowerOfHanoi(n, 'A', 'C', 'B')

# Contributed By Harshit Agrawal
```

The minimum number of moves required to solve the Tower of Hanoi is determined by the following formula: 2^n - 1. Therefore, the time complexity is 0(2^n). Thus, if 64 disks must be moved, this would take 2^64 - 1 seconds, equivalent to 584,542,046,090.6263 years (365.25 days per year). Fun fact: The sun has only 7 billion years to exist before it goes supernova, at which point the world would cease to exist. The application can only move a realistic amount of discs; otherwise, the program will never end, or the system will crash. As a result, it has negative aspects to the security since the attacker can use this weakness and cause a crash or other extensive problems for the system (Khanacademy, 2022).

Regex

The UK postcode system consists of a string that contains a number of characters and numbers – a typical example is ST7 9HV (this is not valid – see below for why). The rules for the pattern are available from idealpostcodes (2020).

Create a python program that implements a regex that complies with the rules provided above – test it against the examples provided.

Examples:

M1 1AA
M60 1NW
CR2 6XH
DN55 1PT
W1A 1HQ
EC1A 1BB

How do you ensure your solution is not subject to an evil regex attack?

→ **Solution on GitHub!** **https://github.com/gicanon/regexPostcode**

Regex command: "^[A-Z]{1,2}[0-9R][0-9A-Z]? [0-9][ABD-HJLNP-UW-Z]{2}$

Evil Regex is a regex pattern which can get stuck on manipulated input. The attacker injects the evil Regex in order to make the system vulnerable. Evil Regex contains a group with repetitions. Examples:

- (a+)+
- ([a-zA-Z]+)*
- (a|aa)+
- (a|a?)+
- (.*a){x} for x \> 10

By entering the input aaaaaaaaaaaaaaaaaaaaaaaa! The system is vulnerable and can get stuck with the Regex above (Weidman, 2022). Hence, it is not subject to an evil regex attack.

References:

Khanacademy, 2022. Towers of Hanoi. Available from: https://www.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi [Accessed 13 October 2022].

Weidman, A. (2022) *Regular expression Denial of Service - ReDoS Author: Adar Weidman.* Available from: https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS [Accessed 11 October 2022].