Hackathon 1 (7th Jan 2022)

Problem 1 (trivial)

Pseudocode:

- Input: $n, a, b \in \mathbb{N}$. Assume without loss of generality that $a < b$.
- Let $i \leftarrow a$ and $sum \leftarrow 0$.
- While $i < b$ do the following
    - if a divides i **or** b divides i then $sum \leftarrow sum + i$
- Print sum

Problem 2 (easy)

Pseudocode:

- Input: $a, b \in \mathbb{N}$ given as two integers separated by a space.
- Let g be the gcd of a and b.
- Let $p \leftarrow a/g$ and $q \leftarrow b/g$.
- Print p/q

Problem 3 (Challenging)

Main Idea: We need to keep track of the opening braces, and "cancel" them out as we see the correct type of closing brace. More precisely, when we see a closing brace, we need to check the *most recent* opening brace and match them. Any mismatch invalidates the string.

Two main approaches to solve this problem.

Apporoach 1 (Fast and Modern, with Stack simulation):

- Input: A string str formed by the following characters: `(`, `)`, `[`, `]`, length at most 20.
- Let $strStack$ denote a character array of length 21. Imagine this string to grow from bottom to top. The 0th element is the "bottom".
- Let _strStack[0]$\leftarrow$ 'B', and $top \leftarrow 0$. Here 'B' denotes bottom.
- For $i \leftarrow 0$ to the end of str
    - If $str[i]$ is an opening brace, then
        - $top \leftarrow top + 1$
        - $strStack[top] \leftarrow str[i]$
    - Else (we have a closing brace at $str[i]$)
        - If $strStack[i]$ has opening brace that matches $str[i]$ then
            - $top \leftarrow top - 1$
        - Else **Return 0** since there is a mismatch, or strStack is depleted.
- If $top \leftarrow 0$ then **Return 1** since everything matches perfectly.
- Else **Return 0**

Approach 2 (Classic, pencil on paper/single tape Turing machine):

- Input: A string str formed by the following characters: `(`, `)`, `[`, `]`, length at most 20.
- For $i \leftarrow 0$ to the end of str do the following
    - If $str[i]$ is a closing brace then
        - For $j \leftarrow (i-1)$ to 0 do:
            - If $str[j]$ is a closing brace, **Return 0**
            - Else if $str[j]$ is an opening brace of wrong type, **Return 0**
            - Else If $str[j]$ is an opening brace of the same type as $str[i]$, then
                - $str[i] \leftarrow$ 'X'
                - $str[j] \leftarrow$ 'X' We "cross out" this matching pair.
                - break out of this loop.
- If the entire string has been crossed out with 'X', then **Return 1**

Remarks:

Remark 1:

Approach 1 and 2 have their pros and cons:

- Approach 1
    - Pros: Does not use nested loops.
    - Cons: Uses extra space, as much as the length of input.
- Approach 2:
    - Pros: No extra space. All computation directly on input string.
    - Cons: Input string is destroyed during processing. Nested loop takes more time.

Remark 2:
- Both approaches generalize to any number of types of braces. The idea of matching pairs correctly is not restricted to just two types of braces.

Remark 3:

- I called the first approach "modern". There is nothing modern about the idea! I called it modern only because on modern day systems, memory space is not a big concern. Running time is. So although approach 1 uses more space, it is the preferred approach on modern systems.

Remark 4:

- On a lighter note, should the second approach be called 'modern' since we literally "cancel" things out like "cancel culture" of today?